A Quantum Circuit to Speed-up the Cryptanalysis of Code-based Cryptosystems

Simone Perriello[1[0000−0001−9656−7252]], Alessandro Barenghi[1[0000−0003−0840−6358]], and Gerardo Pelosi[1[0000−0002−3812−5429]]

Department of Electronics, Information and Bioengineering - DEIB
Politecnico di Milano, 20133 Milano, Italy
Email: simone.perriello@polimi.it, alessandro.barenghi@polimi.it, gerardo.pelosi@polimi.it

Abstract. The significant interest in cryptographic primitives providing sound security margins when facing attacks with quantum computers is witnessed by the ongoing USA National Institute of Standards and Technology Post-quantum Cryptography Standardization process. Sound and precise evaluation of the amount of computation required to break such cryptographic primitives by means of quantum computers is required to be able to choose the cryptosystem parameters.
We present a full description of a quantum circuit to accelerate the computation of the solution of the Information Set Decoding problem , which is currently the best known non-structural attack against code-based cryptosystems. We validate our design running it on small instances of error correction codes, which allowed a complete validation on the AtoS QLM quantum computer simulator. We detail the circuit accelerating the exponential complexity search phase in the Lee and Brickell variant of the ISD solver, and provide its computational complexity for cryptographically relevant parameters taken from the third round candidates in the USA post-quantum standardization process.

Keywords: Post-Quantum Cryptography, Code based cryptography, Information Set Decoding

1 Introduction

The overwhelming majority of widely employed asymmetric cryptographic primitives rely on the assumption that factoring large integers or solving discrete logarithms in cyclic groups with large prime order are infeasible tasks. The most efficient algorithms for integer factoring and the computation of discrete logarithms in numerical cyclic groups with prime order, the *general number field sieve* and the *index calculus*, run in sub-exponential computation time. However, in a pioneering work in 1994, Peter Shor formulated an algorithm running on a quantum computer that computes integer factorization in polynomial time [25]. The same algorithm can be adapted to solve the discrete logarithm problem on any cyclic group [24].

Given the significant efforts put in the construction of large scale quantum computers, designing quantum-computing resistant asymmetric primitives has become a pressing need. This need is also witnessed by the USA National Institute of Standards and Technology (NIST) standardization effort which has begun in November 2017 and has recently entered its third round phase [19], selecting a portfolio of (post-quantum) cryptosystems withstanding cryptanalysis with quantum computers.

Among the mathematical trapdoors available to build a post-quantum cryptographic primitive, cryptosystems based on linear codes represent a prime candidate, with three candidates present in the third round of the NIST standardization effort. The hard problem underlying code-based cryptosystems is either decoding an error affected codeword of a random linear code, or finding an error vector corresponding to a syndrome of a random linear code. Such problems were shown to be NP-hard in [6], and were employed to build two families of cryptosystems, namely McEliece's [13] and Niederreiter's [20]. Both of them build the trapdoor function out of a non-random, efficiently decodable code, constituting the private key of the cryptosystem, and out of an obfuscated representation of the same code, made indistinguishable from a completely random one, which acts as the public key.

The most effective cryptanalytic tool, agnostic to the choice of the non-random efficiently decodable code, is the Information Set Decoding (ISD) technique, proposed by Prange [23]. The original proposal was refined with improvements on the computation complexity, albeit with diminishing rewards over time [3]. The cumulative impact of all the improvements has indeed a small repercussion in the choice of the key size [8]. A crucial point in designing cryptosystem parameters to provide a predetermined computational security margin is to evaluate the computational complexity of the ISD algorithms considering the potential speedups that can be provided by a quantum computer. Currently, only studies providing asymptotic costs as a function of the cryptosystem parameters (e.g., key sizes) are present for quantum accelerated ISDs [7,14]. While asymptotic bounds help to understand the complexity trend, providing finite-regime estimates of the computation demands of cryptanalysis of code-based cryptosystems would allow to derive a more precise estimate of their parameters (e.g., key sizes). In this work, we describe a quantum circuit to accelerate the Lee and Brickell approach to ISD [16], which improves over [23]. The choice of the Lee-Brickell variant allows us to proceed with a quantum circuit that does not need any quantum RAM element, but only standard quantum gates. In a related work [22], the authors propose a complete circuit to implement a quantum version of [23].

Contribution. The aim of this paper is to provide first concrete measures to be used in the evaluation of the complexity parameters required by the NIST post-quantum cryptography program. To the best of our knowledge, this is the first detailed implementation of a quantum circuit that aims to speed up an ISD algorithm. We design the quantum circuit to speed up the exponential time search-phase of the Lee and Brickell (LB) algorithm, while leaving the polynomial

time input preparation to a classical controller. We analyze the circuit complexity of our solution, and validate the soundness of the approach by simulating the circuits with the Atos Quantum Learning Machine (QLM) [2]. We report results on the cryptographically relevant parameters proposed for two code-based cryptosystems included in the third round of the NIST post-quantum standardization initiative, showing gains between $2^{10}\times$ and $2^{20}\times$ with respect to a purely classic ISD implementation.

2 Background

In this section, we recall the basic notions underlying code-based asymmetric cryptography, the ISD strategy and provide background information on quantum computing and Grover's algorithm.

2.1 Code based cryptosystems

An $[n,k]$ binary linear error correcting code C is a k-dimensional vector subspace of $\mathbb{F}_2^n$, whose n-length (column) vectors are called codewords. Codes are designed in such a way that the minimum Hamming distance between any two codewords is at least d, a value known as *code distance*. Starting from a k-bit long input message, $\boldsymbol{m}\in\mathbb{F}_2^k$, the objective of the code is to transform m into a codeword $c\in C$ by adding to it $r=n-k$ redundant bits, a process known as *encoding*.

C can be completely characterized by a generator matrix $\boldsymbol{G}\in\mathbb{F}_2^{k\times n}$ such that $C=\{\boldsymbol{c}\in\mathbb{F}_2^n : \boldsymbol{c}=\boldsymbol{G}^{\mathrm{T}}\boldsymbol{m}\}$. In other words, a codeword $c\in C$ is obtained through a linear combination of the rows of $\boldsymbol{G}$. As an alternative to $\boldsymbol{G}$, a linear code can be described using the so-called parity-check matrix $\boldsymbol{H}\in\mathbb{F}_2^{r\times n}$ as $C=\{\boldsymbol{c}\in\mathbb{F}_2^n : \boldsymbol{H}\boldsymbol{c}=0\}$, with $\boldsymbol{H}\boldsymbol{G}^{\mathrm{T}}=\boldsymbol{0}_{r\times k}$.

The process of removing the r redundant bits of information from a (possibly erroneous) codeword is called *decoding*. Specifically, the *codeword decoding* problem consists in retrieving the original message $\boldsymbol{m}$ starting from a corrupted codeword $\boldsymbol{y}=\boldsymbol{c}+\boldsymbol{e}$, where $\boldsymbol{c}\in C$ and $\boldsymbol{e}$ is an $n\times 1$ binary (column) vector with Hamming weight $\mathrm{WT}(\boldsymbol{e})=t$, $0<t<\lfloor\frac{d}{2}\rfloor$. To this end, we can equivalently recover either $\boldsymbol{c}$ or $\boldsymbol{e}$. To recover the error $\boldsymbol{e}$, many algorithms use the so-called *syndrome*, i.e. the $r\times 1$ (column) vector $\boldsymbol{s}$ defined as $\boldsymbol{s}=\boldsymbol{H}\boldsymbol{y}=\boldsymbol{H}(\boldsymbol{c}+\boldsymbol{e})=\boldsymbol{H}\boldsymbol{e}$. This problem is known as *syndrome decoding*.

If $\boldsymbol{H}$ is randomly chosen among the ones in $\mathbb{F}_2^{r\times n}$ with rank k, both the codeword decoding problem and the syndrome decoding problem are NP-hard [6]. To build a sound cryptosystems, we can therefore start from a non-random and efficiently decodable code with parity-check matrix $\boldsymbol{H}'$, and then conceal it through a multiplication with a random non-singular matrix $\boldsymbol{S}$, yielding $\boldsymbol{H}=\boldsymbol{S}\boldsymbol{H}'$. The random-looking matrix $\boldsymbol{H}$ can then be used by anyone to send a message over a public channel: it suffices to encode the message as an error vector $\boldsymbol{e}$ of weight t, compute its syndrome and send it. An attacker trying to recover the original message $\boldsymbol{e}$ is then forced to find the solution to the syndrome decoding problem for an apparently random parity-check matrix $\boldsymbol{H}$.

2.2 Information Set Decoding (ISD)

The state-of-the-art method to efficiently solve the *syndrome decoding* problem is based on ISD techniques. The underlying intuition, due to Prange in [23], is to see the syndrome $\boldsymbol{s}$ as obtained by the sum of the t columns of $\boldsymbol{H}$ indexed by the bit-positions in the error vector $\boldsymbol{e}$ containing a term equal to one. To find out the bit-positions in $\boldsymbol{e}$ corresponding to an asserted bit, Prange's idea is to guess k bit-positions in $\boldsymbol{e}$ corresponding to a zero-bit. After that, by using a permutation matrix $\boldsymbol{P}_{n\times n}$, Prange's algorithm continues by packing all the columns of $\boldsymbol{H}$ indexed by the said k positions to the leftmost part of $\boldsymbol{H}$, obtaining a new matrix $\hat{\boldsymbol{H}} = \boldsymbol{HP}$. The next step brings $\hat{\boldsymbol{H}}$ in reduced row echelon form, simultaneously finding a linear transformation represented by an $r \times r$ non-singular matrix $\boldsymbol{U}$, such that $\boldsymbol{U}\hat{\boldsymbol{H}} = [\boldsymbol{V}_{r\times k} \mid \boldsymbol{I}_r]$. The leftmost matrix $\boldsymbol{V}_{r\times k}$ is a random-looking matrix, while $\boldsymbol{I}_r$ is an identity matrix with size r. The procedure fails if the reduced row echelon form cannot be obtained, i.e., whenever the rightmost $\boldsymbol{U}\hat{\boldsymbol{H}}$ has a singular $r \times r$ submatrix on its right instead of an identity matrix. In this case, we guess another permutation matrix $\boldsymbol{P}_{n\times n}$ and we recompute the reduced row echelon form.

Observing that $\boldsymbol{s} = \boldsymbol{He} = (\boldsymbol{HP})\boldsymbol{P}^{-1}\boldsymbol{e} = \hat{\boldsymbol{H}}(\boldsymbol{P}^{-1}\boldsymbol{e})$, we have that $\bar{\boldsymbol{s}} = \boldsymbol{Us} = \boldsymbol{U}\hat{\boldsymbol{H}}\boldsymbol{P}^{-1}\boldsymbol{e} = [\boldsymbol{V}_{r\times k} \mid \boldsymbol{I}_r]\boldsymbol{P}^{-1}\boldsymbol{e}$. The permutation is expected to pack in the r rightmost columns of $\boldsymbol{H}$ all its columns indexed by the bit-positions of $\boldsymbol{e}$ corresponding to an asserted bit. As a consequence, also the vector $\boldsymbol{P}^{-1}\boldsymbol{e}$ will have its t asserted bits in its r trailing elements, i.e.: $\boldsymbol{P}^{-1}\boldsymbol{e} = [\boldsymbol{0}_{k\times 1}|\boldsymbol{e}'_{r\times 1}]$. For this reason, $\bar{\boldsymbol{s}}$ can be seen as the syndrome of the permuted error $\boldsymbol{P}^{-1}\boldsymbol{e}$ through the permuted and row echelon reduced parity-check matrix $\boldsymbol{U}\hat{\boldsymbol{H}} = [\boldsymbol{V}_{r\times k} \mid \boldsymbol{I}_r]$, i.e.: $\bar{\boldsymbol{s}} = [\boldsymbol{V}_{r\times k} \mid \boldsymbol{I}_r][\boldsymbol{0}_{k\times 1}|\boldsymbol{e}'_{r\times 1}] = \boldsymbol{e}'_{r\times 1}$. Indeed, in the case that the chosen permutation packs k positions of $\boldsymbol{e}$ corresponding to unasserted bits on its top part, then the multiplication between the syndrome and $\boldsymbol{U}$ produces the non-zero part of the permuted error vector itself. To test if the permutation $\boldsymbol{P}$ actually packed k-zero positions of the error in the first k rows, Prange's algorithm checks if the Hamming weight of $\bar{\boldsymbol{s}}$ is equal to the one of the error vector, t. In this case, the error vector can be retrieved by calculating $\boldsymbol{P}[\boldsymbol{0}_{k\times 1}|\bar{\boldsymbol{s}}] = \boldsymbol{P}[\boldsymbol{0}_{k\times 1}|\boldsymbol{e}'_{r\times 1}] = \boldsymbol{e}$. If the test for the permutation correctness fails, the algorithm restarts, picking another random permutation and repeating the reduced-row-echelon form computation and the test on $\bar{\boldsymbol{s}}$. The complexity of Prange's algorithm can be derived by observing that it is a Las Vegas algorithm, that is, a randomized algorithm which always produce a correct result (i.e., the value of $\boldsymbol{e}$ or it notifies about a *failure*) in a finite running time. The expected runtime of the algorithm depends on the expected number of times, $\mathcal{N}$, it must guess the permutation matrix before succeeding, as well as on the dimensions of $\boldsymbol{H}$, n and r, and the weight t of the error vector $\boldsymbol{e}$, i.e.: $C_{\text{Prange-ISD}}(n,r,t) = \mathcal{N} \cdot C_{\text{Prange-iter}}(n,r)$. Since each matrix guess is independent of the others, $\mathcal{N}$ is computed as $\mathcal{N} = \frac{1}{\Pr_{succ}}$, with $\Pr_{succ}$ being the probability that a single computation succeeds. In Prange's algorithm, $\Pr_{succ}$ is computed by dividing the number of permuted error vector admissible by the hypotheses, i.e. all the vector configurations having all the t error-affected bits in the last r positions, by the total number of possible error vector configurations,

i.e., $\mathrm{Pr}_{succ} = \binom{r}{t}/\binom{n}{t}$. An additional factor that makes a single iteration of the algorithm fail is the probability that the procedure computing the reduced row echelon form of $\hat{\boldsymbol{H}}$ does not succeed. However, the probability that a random $r \times r$ binary matrix is non-singular is $\prod_{i=1}^{r}(1-\frac{1}{2^i})$ [3], a factor quickly converging to ≈ 0.2887 for increasing values of r. This contribution adds a constant factor of about 4 to the number of repetitions $\mathcal{N}$ be done by Prange's ISD. As a consequence, the prevailing cost in a single repetition of Prange's algorithm is the computation of the row-echelon form reduction of $\hat{\boldsymbol{H}}$, which takes $\mathrm{C}_{\mathrm{RREF}} = \frac{3nr^2}{4} + \frac{nr}{4} - \frac{n}{2} + \frac{3r^2}{4} - \frac{r}{2}$ bit operations via a simple Gaussian elimination. Finally, to compute $\boldsymbol{U}\boldsymbol{s}$ and to check the weight of $\bar{\boldsymbol{s}}$, the algorithm requires $O(r^2)$ and $7r+\log_2(r)$ bit operations.

Lee and Brickell's ISD. The exponential contribution to the algorithmic complexity of Prange's ISD comes from the number of iterations that have to be computed, which is exponential in n and t. The ISD algorithm proposed by Lee and Brickell [16] reduces the average number of iterations, at the cost of adding a computation with exponential complexity for each iteration.

The main idea of this optimization is to tolerate the presence of $p \geq 1$ asserted bits of the permuted error, $\boldsymbol{P}^{-1}\boldsymbol{e}$, in its leftmost k bit-positions adding all the possible $\binom{k}{p}$ choices of columns from the $\boldsymbol{V}$ matrix before performing the check on the weight of $\boldsymbol{U}\boldsymbol{s}$. To do so, consider $\boldsymbol{P}^{-1}\boldsymbol{e} = [\boldsymbol{e}'_{k\times 1}|\boldsymbol{e}''_{r\times 1}]$, with $\mathrm{WT}(\boldsymbol{e}') = p$ and $\mathrm{WT}(\boldsymbol{e}'') = t-p$. Denoting as $v_j, 0 \leq j < k$, the j-th column of the submatrix $\boldsymbol{V}$ of $\hat{\boldsymbol{H}}$, $\boldsymbol{U}\hat{\boldsymbol{H}} = [\boldsymbol{V}_{r\times k} \mid \boldsymbol{I}_r]$, we can now rewrite $\bar{\boldsymbol{s}} = \boldsymbol{U}\boldsymbol{s}$ as $\bar{\boldsymbol{s}} = [\boldsymbol{V}|\ \boldsymbol{I}_r][\boldsymbol{e}'_{k\times 1}|\boldsymbol{e}''_{r\times 1}] = \boldsymbol{V}\boldsymbol{e}'_{k\times 1} + \boldsymbol{e}'' = \boldsymbol{e}'' + \sum_{j\in \mathrm{supp}(\boldsymbol{e}')} v_j$, where $\mathrm{supp}(\boldsymbol{w})$ denotes the set of indexes of the asserted components/bits in $\boldsymbol{w}$. Rearranging the terms, we obtain $\bar{\boldsymbol{s}} - \sum_{j\in \mathrm{supp}(\boldsymbol{e}')} v_j = \boldsymbol{e}''$, which states that, if p asserted components were present in the portion of the permuted binary error vector multiplied by $\boldsymbol{V}$, the weight $\mathrm{WT}\left(\bar{\boldsymbol{s}} - \sum_{j\in \mathrm{supp}(\boldsymbol{e}')} v_j\right) = \mathrm{WT}\left(\left(\bigoplus_{j\in \mathrm{supp}(\boldsymbol{e}')} v_j\right) \oplus \bar{\boldsymbol{s}}\right)$ corresponds to the number of asserted components of the permuted error portion multiplied by $\boldsymbol{I}$, that is $t-p$.

Given a parity-check matrix $\boldsymbol{H}$ and a syndrome $\boldsymbol{s}$, the Lee and Brickell's variant of the ISD thus requires each iteration of the algorithm to compute the reduced row echelon form of the permuted parity-check matrix $\hat{\boldsymbol{H}}$ deriving $\boldsymbol{U}$ such that $\boldsymbol{U}\hat{\boldsymbol{H}} = [\boldsymbol{V}|\boldsymbol{I}_r]$, compute $\bar{\boldsymbol{s}} = \boldsymbol{U}\boldsymbol{s}$ and then test, for all the $\binom{k}{p}$ possible sums of p columns of $\boldsymbol{V}$, if adding them to $\bar{\boldsymbol{s}}$ yields a vector with Hamming weight equal to $t-p$. If the check is successful, then the positions of the asserted bits in the permuted error vector portion $\boldsymbol{e}''$ are derived from the positions of the asserted bits in vector resulting from the sum of $\bar{\boldsymbol{s}}$ and the p columns of $\boldsymbol{V}$ that have been selected, while the positions of the asserted bits in $\boldsymbol{e}'$ are derived from the indexes of the columns of $\boldsymbol{V}$ that have been added to the syndrome $\bar{\boldsymbol{s}}$.

The computational cost of a single iteration of the Lee and Brickell algorithm is $C_{\text{Lee-Brickell-iter}}(n,r,p) = \mathrm{C}_{\mathrm{RREF}} + O(r^2) + O\left((rp + 7r + \log_2(r))\binom{k}{p}\right)$ bit operations. The last term considers the cost of performing $p-1$ additions of r-bit long vectors for each weight test, plus the linear cost of the weight test itself. Such an increase in the computation cost of a single iteration is traded off for a significant increase

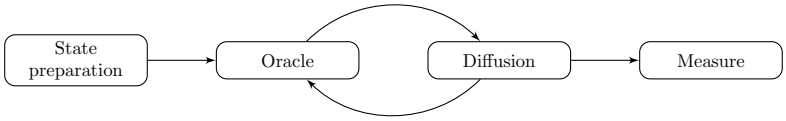


Fig. 1. High level overview of Grover's algorithm to speed up the computation of the zeroes of a generic Boolean function.

in the probability of success of a single iteration (i.e., the guessing of a matrix $\boldsymbol{P}$), which becomes $\Pr_{\mathrm{LB}-succ} = \frac{\binom{k}{p}\cdot\binom{r}{t-p}}{\binom{n}{t}}$. We note that, while p is a free parameter in the Lee-Brickell ISD variant, its asymptotically optimal value was found to be 2 with classic computing, and tested by exhaustive search in [3].

The exponential-time exhaustive search with complexity $O\left(rp\binom{k}{p}\right)$ for the position of the p remaining errors in the permuted error vector is the main candidate to be sped up by a quantum computer. Indeed, the reduced row echelon form computations and the matrix-vector multiplications, required in the remainder of the algorithm, are polynomial-time procedures, which have a negligible impact on the iteration time for cryptographically relevant parameters [9].

2.3 Grover's algorithm

In this work we will rely on the algorithmic framework proposed by Grover [12], in short known as Grover's algorithm. The algorithm finds the value of the input $x^* \in \{0,1\}^n$ such that the Boolean function $f : \mathbb{F}_{2^n} \mapsto \mathbb{F}_2$ evaluates to 1, under the assumption that only a single value x^* exists, in $O(\sqrt{2^n})$ function computations. The classical algorithm, on the other hand, needs $O2^n$ function computation to obtain the result.

Grover's framework, summarized in Figure 1, relies on four steps, of which the second and the third are repeated $O(\sqrt{2^n})$ times: the preparation of a quantum computer input state, the computation of the so-called oracle function, the computation of a diffusion function, and a final measurement stage. The end goal of the algorithm is to obtain x^* with a high probability upon measurement, using as few iterations as possible of the oracle and diffusion steps. We will now detail the description of the first three stages of the Grover framework.

State preparation. The framework proposed by Grover begins by constructing a uniform superposition of all the possible basis states, $|\psi_0\rangle$. Each basis state can be thought as labeled with binary strings corresponding to the elements of the Boolean function f, all with equal amplitudes. Denoting with $\mathbb{D} \subseteq \{0,1\}^n$ the domain of function f, with $|\mathbb{D}| = d$, we want to have an input quantum state described as

$$|\psi_0\rangle = \frac{1}{\sqrt{d}} \sum_{x \in \mathbb{D}} |x\rangle = \alpha_r^* |x^*\rangle + \alpha_r \sum_{x \neq x^*} |x\rangle \tag{1}$$

where α_r and α_r^* are the amplitudes associated to states $x \neq x^*$ and x^* respectively. At this point we have $\alpha_r = \alpha_r^* = \frac{1}{\sqrt{d}}$. In the original article by Grover,

and in the vast majority of the literature, such superposition contains all the basis states, as the Boolean function admits any Boolean string with a matching length as input.

Oracle function computation. The computation of the oracle function in Grover's framework aims at identify the basis state corresponding to x^*, the value we are looking for. One computation doing so acts on $n+1$ qubits, where the first n encode the input to f and the last one is initialized to 0 and maps $|x,0\rangle \mapsto |x,f(x)\rangle$, in which the comma is used to stress the action of the oracle on the last qubit. Concretely, this computation can be performed with a set of quantum gates corresponding to the classical gates required to compute f applied to the first n qubits, adding the result to the last qubit. The computation employed in Grover's algorithm starts from the previous approach and considers what happens if, instead of setting the last qubit to $|0\rangle$, it is set to $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. It can be shown that the computation maps $|x,-\rangle \mapsto (-1)^{f(x)}|x,-\rangle$, effectively storing the information on whether the basis vector is the one corresponding to x^* in the sign of the amplitude of the basis vector itself. The use of the additional result qubit, although convenient for explanation purposes, is not required in practical implementation, and the computation performed is then represented as $|x\rangle \mapsto (-1)^{f(x)}|x\rangle$. Applying this computation to the initial state prepared during the state preparation step $|\psi_0\rangle$ yields the following mapping:

$$\alpha_r^*|x^*\rangle + \alpha_r \sum_{x\neq x^*} |x\rangle \;\mapsto\; -\alpha_r^*|x^*\rangle + \alpha_r \sum_{x\neq x^*} |x\rangle$$

effectively singling out the basis vector representing x^* by changing the sign of its amplitude α_r^*.

Diffusion. The task of the diffusion stage is to build on the output of the oracle stage to produce a superposition in which α_r^* — the amplitude of the vector representing x^* — increases its modulus. To do so, Grover observed that the state after the oracle can be rewritten as

$$|\xi\rangle = -\alpha_r^*|x^*\rangle + \alpha_r \sum_{x\neq x^*} |x\rangle = |\psi_0\rangle - 2\alpha_r^*|x^*\rangle \tag{2}$$

From this, Grover observed that applying the linear transform $\mathrm{S} = 2|\psi_0\rangle\langle\psi_0| - I$ to $|\xi\rangle$, and recalling that we indicate with d the size of the domain of the function f, we obtain:

$$|\psi_1\rangle = S|\xi\rangle = \frac{d-4}{d}|\psi_0\rangle + \frac{2}{\sqrt{d}}|x^*\rangle$$

Therefore, the state $|\psi_1\rangle$ is a non-uniform superposition of all the possible basis states, where the one representing the value x^* has twice the amplitude with respect to a uniform superposition. Concretely, the S transform is computed as PS_0P^{-1}, where S_0 is a transform inverting the sign of the amplitude of the all-0 quantum state, while P is the transform which maps the $|0^n\rangle$ state onto $|\psi_0\rangle$.

Number of iterations. Grover observes that applying again the oracle computation and diffusion procedure to $|\psi_1\rangle$ will keep increasing the amplitude of the desired basis state, while reducing the others. The computation is repeated for a number of times sufficient to make the probability amplitude of the searched state $|x^*\rangle$ grow to a point where measuring the superposition will yield it with high probability. Care must be taken in choosing the right number of iterations since the probability of success does not increase monotonically with the number of iterations. It can be shown indeed that the optimal number of iterations to have a close to 1 chance of observing $|x^*\rangle$ upon measurement is $\approx O(\sqrt{d})$. As shown in [10] we can also use half the number of iterations to have a probability close to 50% to observe $|x^*\rangle$.

Amplitude sign flip subcircuit. The goal of this subcircuit, used in both the oracle and the diffusion stage, is to invert the sign of the amplitude associated to a specific quantum state, and it can be implemented as a multi-controlled Z (MCZ). This gate indeed inverts the sign of the all-1's state, while leaving all other states unchanged. If the MCZ involves m qubits, $m-1$ qubits act as control qubits, while one acts as target. Our goal is therefore to express the state for which we want to change the sign of the amplitude as an all 1-state right before performing the sign flip. For the oracle, this goal requires to translate the $|x^*\rangle$ into an all-1 state on the qubits expressing the result of the function evaluation. On the other hand, in the diffusion stage, S_0 requires a sign inversion of the all-0 state. As a consequence, to be able to use the MCZ, we need to apply an X gate before and after the MCZ in order to have a transform equivalent to S_0.

Another relevant difference between the oracle and diffusion phase is in the involved qubits. While in the oracle the circuit acts on all the qubits storing the result of the oracle computation, in the diffusion the involved qubits are the ones storing the input quantum state superposition. In the plain version of Grover's algorithm those qubits are identical, but it is not necessarily the case.

3 A Quantum Circuit to Speed-up ISD Iterations

In this section, we describe our approach to the acceleration of the exponential-time exhaustive search for the sum of the p columns of $\boldsymbol{V}$ which, added to the transformed syndrome $\bar{\boldsymbol{s}}$, yields a weight $t-p$ vector, as required in the Lee-Brickell ISD. To employ the algorithmic framework proposed by Grover, we recast the problem of finding the correct sum of p columns of $\boldsymbol{V}$ into a Boolean function with a mono-dimensional output. We consider the Boolean function f, $f : \mathbb{D} \to \{0,1\}$, where $\mathbb{D} \subseteq \{0,1\}^k$ is the set of all the weight-p, length-k Boolean vectors. Let $x \in \mathbb{D}, x = (x_0, \ldots, x_{k-1})$; we define f as:

$$f(x_0, \ldots, x_{k-1}) = \begin{cases} 1 & \text{if } \mathrm{WT}\left(\left(\bigoplus_{j \in \mathrm{supp}(\boldsymbol{e}')} v_j\right) \oplus \bar{\boldsymbol{s}}\right) \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

The goal of the algorithm is to find the unique input x^* for which f returns 1. As explained in Section 2.3 we need to create a quantum circuit, the oracle,

that inverts the sign of the amplitude of $|x^*\rangle$, leaving all other quantum states untouched. To exploit the power of quantum computing we additionally want to prepare an equal-weight superposition of all the possible inputs at once.

3.1 Preparing a superposition of all column selections

As per Equation (3), our algorithm expects as input a state which is the superposition of states representing length-k, weight-p Boolean strings. This state, called Dicke state and represented as $\left|D_p^k\right\rangle$, is defined as $\left|D_p^k\right\rangle = \binom{k}{p}^{-\frac{1}{2}} \sum_{\mathrm{WT}(x)=p} |x\rangle\,, \qquad x \in \{0,1\}^k$. To the best of our knowledge, the most efficient quantum circuit to prepare a Dicke state for a generic value of p is the one presented in [18], which is an improvement over the previous best algorithm presented in [5]. The solution proposed in [5] is quite efficient, as it only requires p X gates, and $O(k+kp)$ CNOT gates and RY gates, with an overall depth equal to $O(k)$. The work of [18] improves the previous solution in its cost in CNOT and RY gates to $O(k+p^2)$, while keeping the same circuit depth: we report the detailed cost of generating $\left|D_p^k\right\rangle$ in Table 1.

We recall that the qubits involved in the input preparation will be used in the diffusion stage. As explained in Section 2.3, in the amplitude sign flip stage we have to apply an X gate on the involved qubits before and after the flip. In our algorithm the involved qubits are the ones belonging to the `sel` register.

3.2 Designing Grover's oracle function

We now describe the quantum circuit to compute Grover's oracle function for the Boolean function f. This quantum circuit is split in five stages: i) the selection of a set of columns of $\boldsymbol{V}$ corresponding to the set entries of the Boolean input vector $|x\rangle$, ii) the computation of the actual Boolean sum (xor) of the columns themselves and the syndrome $\bar{\boldsymbol{s}}$, iii) the computation of the Hamming weight of the result of ii), iv) the comparison of the Hamming weight with the integer value $t-p$ and v) the amplitude sign flip if the previous comparison is successful.

In order to leave the input states $|x\rangle \neq |x^*\rangle$ unaltered, as demanded by the oracle model, we have to perform an uncomputation of the oracle circuit before the flip phase, along the lines of the common compute-uncompute paradigm of quantum computing. Since the uncomputation is simply the application of the same gates in reverse order, we do not detail this portion of the circuit.

For the sake of clarity, we will assume that r is a power of two in the description. We will employ a running example where $\boldsymbol{V}$ is a 4×4 binary matrix and the transformed syndrome $\bar{\boldsymbol{s}}$ vector is 4 elements long, with the following values:

$$\boldsymbol{V} = \underbrace{\left.\begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix}\right\} r}_{k} \qquad \bar{\boldsymbol{s}} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

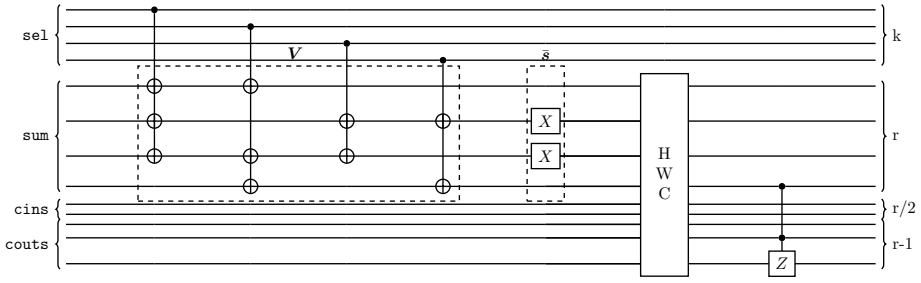


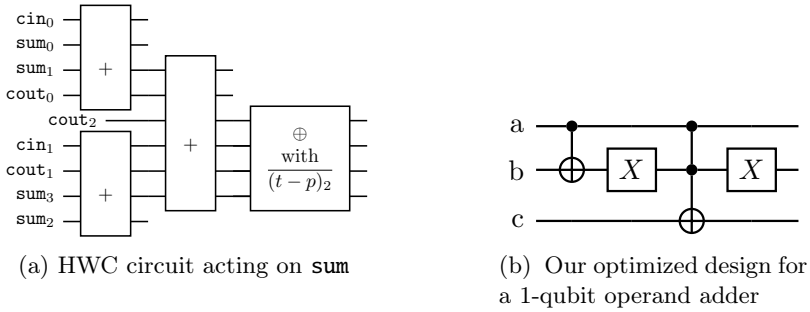
Fig. 2. High level overview of the circuit employed as the Grover oracle.

Figure 2 depicts the instance of the proposed circuit corresponding to the running example. The circuit acts on four quantum registers: `sel` (k qubit wide), containing the input superposition; `sum` (r qubit wide), containing the superposition of the addition of the columns of $\boldsymbol{V}$ after stage ii); `cin` ($\frac{r}{2}$ qubit wide) and `cout` ($r-1$ qubit wide), used by the Hamming weight computation and check (stage iii) and iv)) to store carries.

Columns and syndrome addition. We compute in superposition all the possible Boolean sums (xor) of p columns out of k from the matrix $\boldsymbol{V}$, storing them in the `sum` register. We note that the matrix $\boldsymbol{V}$ is known to the classical controller circuit driving the quantum computer at each repetition of the ISD procedure, and can therefore be considered as fixed from the quantum computer standpoint.

To compute the sum we employ each i-th qubit of the `sel` register, set to the $\left|D_p^k\right\rangle$ state, to drive a group of CNOT gates, one for each set term in the i-th column of $\boldsymbol{V}$. Considering our running example, the first (topmost in Figure 2) qubit of `sel` controls three CNOT gates acting on the first three bits of `sum`, since $\boldsymbol{v}_0$, the first column of $\boldsymbol{V}$, is equal to $[1110]^T$. Performing the addition of the columns of $\boldsymbol{V}$ in this fashion requires an amount of CNOT gates equal to the number of set terms in the matrix $\boldsymbol{V}$. Since $\boldsymbol{V}$ is the result of a reduced row echelon form computation on a random matrix, we expect that, on average, half of its elements will be equal to 1, giving therefore an average of $\frac{rk}{2}$ CNOT (and a worst case of rk CNOTs). From a circuit depth standpoint, we note that, if $k \geq r$ (as it is extremely common in code-based cryptography) it is possible to schedule the action of the rk CNOTs of the worst case in a circuit with depth at most k. Indeed, such a scheduling is possible as there are k controlling lines which pilot r CNOTs each. It is thus possible, at each gate layer, to run CNOTs with both the controlling and controlled bits different from each other.

Finally, the circuit needs to add (xor) the transformed syndrome $\bar{\boldsymbol{s}}$ to the contents of the `sum` register, regardless of the choice of added columns from $\boldsymbol{V}$. This is done via a set of X gates, applied to the qubits in `sum` corresponding to set terms in the $\bar{\boldsymbol{s}}$ vector. In our running example, the X gates are applied to the second and third qubits of `sum`, as the transformed syndrome value $\bar{\boldsymbol{s}}$ is [0110]. We observe that the syndrome addition will require at most r X gates, and $\frac{r}{2}$ gates on average, that can be interleaved with the previous column additions.



(a) HWC circuit acting on `sum`

(b) Our optimized design for a 1-qubit operand adder

Fig. 3. (a)) shows the two subcircuits used for Hamming weight compute and check on the `sum` register for the running example. The computation result is stored on $\texttt{cout}_2$, $\texttt{cout}_1$ and $\texttt{sum}_3$. Afterward, it is compared against the one's complement of the Boolean representation of the constant value $(t-p)_2$. (b)) shows our construction of an adder working with two 1-qubit register. The sum of the input value a and b is put in c and b, where c is the most significant bit.

Computing the Hamming weight. After completing the sum of the columns of $\boldsymbol{V}$ and the transformed syndrome $\bar{\boldsymbol{s}}$, the oracle circuit computes the Hamming weight of the result (stage iii)). This is depicted in Figure 2 as the Hamming Weight Check gate, (HWC for short), acting on `sum`, `cin` and `cout`, while Figure 3a reports the full HWC circuit for our running example. Our approach to computing the Hamming weight stems from the classical one employing a logarithmic depth adder tree. We compute the superposition of the values of the Hamming weights of the r-sized `sum` register using an adder tree of depth $\log_2(r)$. The first layer of the adder tree has only 1 qubit operand adders that store their carry-out on a qubit from `cout`. Later layers of the adder tree use the output of early layers to calculate the sum, until we obtain a single sum as output of the last layer. To be able to use quantum adders in our circuit, we used Cuccaro proposal of [11], a reversible variant of the classical ripply carry adder. The Cuccaro adder performs a partially in-place addition by storing the sum of its two input registers `x` and `y`, containing the binary representation of values a and b respectively, on `y`. It also uses an additional qubit for the carry-in and one more to store the carry-out. The carry-in qubit is always restored to its initial state at the end of the adder circuit and can thus be reused by all adders. However, to run all the adders at the same layer in parallel, we employ a total number of carry-in qubits exactly equal to the number of adders at the first layer.

To compute the number of gates required by the HWC subcircuit, we observe that a Cuccaro adder acting on two n-qubit input registers requires $2n-1$ CCNOT gates, $5n+1$ CNOT gates and $2n$ X gates, with a depth of $2n+6$. For $n=1$, we enhanced Cuccaro proposal by implementing an adder (Figure 3b) requiring only 1 CCNOT, 1 CNOT and 2 X gates with depth 4. The adder tree circuit requires $\log_2(r)$ layers. The i-th layer employs $r/(2^i)$ adders, for a total of $\sum_{i=1}^{\log_2(r)} \frac{r}{2^i} = r-1$ adders. Consequently, $\frac{r}{2}$ carry-in and $r-1$ carry-out ancillary

Table 1. Number of quantum gates and depth as a function of Lee-Brickell ISD parameters for the different subcircuits. We need an additional MCZ($\log_2(r)$) gate for the Oracle stage and an MCZ($k-1$) gate for the Diffusion stage.

State preparation	
Cost metric	**Dicke state i)**
X	p
CNOT	$5kp - 5p^2 - 2k$
RY	$4kp - 4p^2 - 2k + 1$
Depth	$\mathcal{O}(k)$

Diffusion	
Cost metric	**v)**
X	$k + 2p$
CNOT	$10kp - 10p^2 - 4k$
RY	$8kp - 8p^2 - 4k + 2$
Depth	$\mathcal{O}(k)$

Oracle

Cost metric	**Column addition ii)**	**Hamming weight compute iii), check iv)**
X	r	$8r - 2\log_2(r) - 2\log_2(t-p) - 6$
CNOT	rk	$9r - 10\log_2(r) - 22$
CCNOT	0	$6r - 4\log_2(r) - 6$
Depth	$\mathcal{O}(k)$	$2\log_2^2 r + 14\log_2(r) - 8$

qubits are required. Each adder belonging to the i-th layer accepts as input two i-qubit strings. We can therefore compute the overall number of gates required as $\sum_{i=1}^{\log_2(r)} \frac{r}{2^i}$ ADD_COST(i), with ADD_COST(i) denoting the gate cost of a single adder accepting as inputs two i-qubit strings. The summation expansion gives the gate-count shown in Table 1. The global depth can be computed by summing together the depth of just a single adder per level, since at the same level all the adders act on distinct qubits and can therefore be run in parallel.

Since Cuccaro's adder reuses one of the input registers to store part of the sum, at stage i the result is stored on $i+1$ qubits. Therefore, in the final stage, the overall sum is stored on $\log_2(r)+1$ qubits, that we will denote as `hw`.

Checking the Hamming weight value and flipping stage. The final stage of the oracle function for Grover's algorithm compares the obtained Hamming weight values in superposition with the target weight of $t-p$ (stage iv)) and flips the amplitude of the states where a match occurs (stage v)). For this reason, we perform a xor operation between the `hw` register, containing the superposition of Hamming weight computed in phase iii), and the one's complement of the Boolean representation of the constant value $(t-p)_2$. In this way, if a certain state is such that WT(`hw`) $= t-p$, all the qubits belonging to `hw` will be in state $|1\rangle$. As a result, we can apply the multi-controlled Z (MCZ) to invert the sign of the amplitude of $|x^*\rangle$. This process requires only X gates. Indeed, in the worst case, $p = t$ and the value of the addend is the all-ones string represented on $\log_2(r)+1$ qubits, a quantity smaller than r. As a result, we need $\log_2(r) + 1 - \log_2(t-p)$ X gates to carry out the xor operation. If the output of stage iii) is a state containing the binary representation of $t-p$, the said basis state will have, inside the `hw` register, an all-ones bitstring. We finally employ `hw` in a multicontrolled Z gate to perform the amplitude sign flip, as described in Section 2.

A noteworthy point to analyze is how the n-controlled Z gate is translated into a sequence of elementary gates. The approach used in [4] requires $2^n - 2$ CNOT gates, $2^n - 1$ gates $\in SU(2)$. Therefore, the number of gates increases exponentially with the number of control qubits, producing a huge impact on the resource consumption, without however impacting adversely the number of qubits required. On the other hand, in [21], a simpler approach was presented, requiring n additional qubits and an equal number of CCNOT, plus 1 CZ. The total depth of this decomposition is $O(n)$.

4 Experimental Evaluation

To validate the functionality of the proposed circuit for the Grover oracle function and the entire resulting Grover we employed codes smaller than the ones employed in cryptographic systems, to allow their simulation on classic hardware. We chose the original Hamming code $(n, k, d) = (7, 4, 3)$, which is able to correct $t = 1$ error bit, and random codes up to $n = 23$ and $k = 12$ able to correct more than one error bit.

The validation of our Grover-accelerated Lee-Brickell search was done precomputing a pair $(\boldsymbol{V}, \bar{\boldsymbol{s}})$ matching Lee-Brickell requirements on the weights of the two portions of the permuted error $\boldsymbol{P}^{-1}\boldsymbol{e}$. We employed this pair $(\boldsymbol{V}, \bar{\boldsymbol{s}})$ to simulate the execution of the corresponding quantum circuit on the Atos Quantum Learning Machine [2] simulator. After successfully validating the soundness of our circuit, we simulated the execution of the entire ISD procedure, including the classical random guessing of the required permutation.

To evaluate the computational complexity advantage of our solution with cryptographically relevant parameters, we consider the parameter sets for each of the NIST security levels for both the Classic McEliece [8] cryptosystem, the finalist in the NIST Post quantum standardization effort among code based cryptosystems, and Bike [1], one of the alternate candidates. We left out the HQC [17] cryptoscheme from our analysis since its specification consider a raw asymptotic limit, independent from the ISD technique used. The NIST security levels are defined as the computational effort of breaking one of the three AES variants, and therefore correspond to a computational effort of about 2^{128} (level 1), 2^{192} (level 3) or 2^{256} (level 5) AES encryptions. Table 2 reports the number of gates, split by kind, and the number of qubits needed to build the Grover oracle according to our design. Table 2 also reports the optimal value of p for the Lee-Brickell ISD with our technique. The first noteworthy point is that the optimal value of p, obtained through an exhaustive design space exploration, is higher than the one for the classic counterpart, i.e. $p = 2$. This is a result of speeding up the exponential search of the Lee and Brickell ISD, therefore comparatively reducing the amount of computation to be done for a given choice of the p parameter.

The second point is that our design requires a relatively small amount of qubits, at most 2^{17}, making our design plausible even in the scenario where quantum computers with millions of qubits are not available. We notice also

Table 2. Number of gates, qubits and depth required to build the proposed quantum circuit. Figures are relative to the value of p minimizing the total number of gates required.

Scheme	NIST level	p	Grover Iter.s	Grover Depth	Grover Qubits	Grover Gate count X	CNOT	CCNOT	RY	CZ	Tot.
BIKE	1	3	2^{18}	2^{34}	2^{15}	2^{36}	2^{45}	2^{35}	2^{36}	2^{19}	2^{45}
	3	3	2^{20}	2^{37}	2^{16}	2^{38}	2^{49}	2^{37}	2^{39}	2^{21}	2^{49}
	5	3	2^{21}	2^{38}	2^{17}	2^{40}	2^{51}	2^{40}	2^{40}	2^{22}	2^{51}
McEliece	1	13	2^{57}	2^{71}	2^{12}	2^{71}	2^{78}	2^{70}	2^{75}	2^{58}	2^{78}
	3	15	2^{67}	2^{81}	2^{12}	2^{81}	2^{89}	2^{81}	2^{85}	2^{68}	2^{89}
	5	27	2^{124}	2^{139}	2^{13}	2^{139}	2^{147}	2^{138}	2^{144}	2^{125}	2^{147}

Table 3. Comparison between our hybrid Lee-Brickell algorithm, the classical Lee-Brickell algorithm and the full-quantum Prange algorithm proposed in [7].

Scheme	NIST Level	Proposed hybrid ISD (total) p	Quant. gates	Class. gates	Depth	Qubits	Classic ISD p	Asymp. Time	Quantum [7] Asymp. Gates	Asymp. Qubits
BIKE	1	3	2^{161}	2^{159}	2^{159}	2^{15}	2	2^{165}	2^{111}	2^{29}
	3	3	2^{228}	2^{225}	2^{225}	2^{16}	2	2^{232}	2^{147}	2^{31}
	5	3	2^{295}	2^{291}	2^{291}	2^{17}	2	2^{299}	2^{182}	2^{32}
McEliece	1	13	2^{152}	2^{106}	2^{145}	2^{12}	2	2^{162}	2^{107}	2^{23}
	3	15	2^{194}	2^{139}	2^{186}	2^{12}	2	2^{206}	2^{129}	2^{24}
	5	27	2^{300}	2^{188}	2^{291}	2^{13}	2	2^{321}	2^{189}	2^{26}

that the circuit depths obtained for the optimal p of the BIKE cryptosystem are below the 2^{40} mark reported by NIST as the most stringent bound for the plausibility of a sequential computation [19].

We note that the depth of our quantum circuit does not depend on the value of p, while the number of gates involved depends on $O(p^2)$. However, the largest contribution of p to the total computational cost of the quantum part of our algorithm, $O\left(\sqrt{\binom{k}{p}}\right)$, is given to the number iterations. Therefore, hardware realizations of quantum computers will only tolerate depths lower than the one required by the optimal choice of p, our algorithm can be re-tuned to adapt.

Furthermore, our approach can be parallelized over multiple separate quantum computers, thanks to the probabilistic nature of the ISD. Indeed, all the iterations of the ISD algorithm can be run in parallel: separate controllers can prepare independent pairs of $(\boldsymbol{V}, \bar{\boldsymbol{s}})$ to be fed to the quantum computers, simply taking care of exploring separate portions of the column permutation space.

Table 3 reports the overall computational cost of our quantum accelerated Lee-Brickell algorithm, compared to the estimated gate counts for its classic counterpart. In order to estimate the number of repetitions of the oracle-diffusion computations, we consider a number of repetitions which allows us to reach a

probability as close as possible to 1 to observe $|x^*\rangle$ upon measurement. It is possible to halve the number of Grover iterations accepting a success probability close to 50%; however, given the total computational effort, a factor of 2 does not have a significant impact.

Table 3 also reports the results of the asymptotic estimates for the number of quantum gates and qubits required by the high level description of a Prange's ISD algorithm provided in [7]. The approach of [7] employs a full-quantum strategy, in which the guess of the information set and the reduced row echelon form computation are also performed by the quantum device. The proposal reports a number of qubits equals to $n^{O(1)}$. Given that we must represent the whole generator matrix $G \in \mathbb{F}_2^{k \times n}$ and the codeword $c \in \mathbb{F}_2^n$, it is reasonable to assume that the cost will be in the order of $O(n^2)$. The estimate for the number of gates provided in [7] is $O(n^3)$ for a single Grover iteration, with a number of iterations equal to $\sqrt{\binom{n}{k}/0.29\binom{n-t}{k}}$. It is worth noting that [7] leaves the input preparation and diffusion stage out of the gate counts, while they require additional quantum gates, and considers only asymptotic gate costs. Further research is therefore needed in order to have closed-form figures for the proposed quantum circuit.

We highlight that a promising direction for future work is the realization of quantum ISD circuits concretizing the approaches of [14, 15]. Indeed, the works evaluate only the expected asymptotic computation costs of two ISD approaches, highlighting that they have the potential to improve on the ones of Prange and Lee-Brickell.

5 Concluding Remarks

We presented a quantum circuit to speed up the execution of the exponential-time portion of the Lee and Brickell's ISD algorithm. Our proposal allows to implement a quantum accelerator for the Lee-Brickell variant of the ISD, keeping the circuit depth within the most stringent bounds pointed out by the USA NIST [19], and a qubit count varying from 2^{13} to 2^{17} for all set of parameters of two finalists of the NIST competition. Our solution gains a factor between $2^{10}\times$ and $2^{20}\times$ in speed with respect to the purely classic implementation, providing a concrete quantification of the number of gates of the quantum circuit. The proposed solution also allows to pipeline the classic and quantum computations hiding the classic controller latency and allows to parallelize the operations on multiple quantum accelerators.

Acknowledgment

The research activity was partially funded by Atos Italia S.p.A. with a research grant on quantum computing.

References

1. Aragon, N., Barreto, P.S.L.M., Bettaieb, S., Bidoux, L., Blazy, O., et al.: BIKE: Bit Flipping Key Encapsulation. https://bikesuite.org
2. Atos: Quantum Learning Machine, https://atos.net/en/solutions/quantum-learning-machine
3. Baldi, M., Barenghi, A., Chiaraluce, F., Pelosi, G., Santini, P.: A finite regime analysis of information set decoding algorithms. Algorithms **12**(10), 209 (2019), https://doi.org/10.3390/a12100209
4. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Physical rev. A **52**(5), 3457 (1995)
5. Bärtschi, A., Eidenbenz, S.J.: Deterministic preparation of dicke states. In: Gasieniec, L.A., Jansson, J., Levcopoulos, C. (eds.) Fundamentals of Computation Theory - FCT 2019, Copenhagen, Denmark, August 12-14, 2019. LNCS, vol. 11651, pp. 126–139. Springer (2019), https://doi.org/10.1007/978-3-030-25027-0_9
6. Berlekamp, E.R., McEliece, R.J., van Tilborg, H.C.A.: On the inherent intractability of certain coding problems (Corresp.). IEEE Trans. Information Theory **24**(3), 384–386 (1978), https://doi.org/10.1109/TIT.1978.1055873
7. Bernstein, D.J.: Grover vs. mceliece. In: Sendrier, N. (ed.) Post-Quantum Cryptography, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. LNCS, vol. 6061, pp. 73–80. Springer (2010), https://doi.org/10.1007/978-3-642-12929-2_6
8. Bernstein, D.J., Chou, T., Lange, T., von Maurich, I., Misoczki, R., Niederhagen, R., Persichetti, E., Peters, C., Schwabe, P., Sendrier, N., Szefer, J., Wang, W.: Classic mceliece: conservative code-based cryptography. https://classic.mceliece.org/nist/mceliece-20201010.pdf (2020)
9. Bernstein, D.J., Lange, T., Peters, C.: Smaller decoding exponents: Ball-collision decoding. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. LNCS, vol. 6841, pp. 743–760. Springer (2011), https://doi.org/10.1007/978-3-642-22792-9_42
10. Boyer, M., Brassard, G., Høyer, P., Tapp, A.: Tight bounds on quantum searching. Fortschritte der Physik: Progress of Physics **46**(4-5), 493–505 (1998)
11. Cuccaro, S.A., Draper, T.G., Kutin, S.A., Moulton, D.P.: A new quantum ripple-carry addition circuit. arXiv quant-ph/0410184 (2004)
12. Grover, L.K.: A Fast Quantum Mechanical Algorithm for Database Search. In: Miller, G.L. (ed.) Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996. pp. 212–219. ACM (1996), https://doi.org/10.1145/237814.237866
13. J. Mceliece, R.: A Public-Key Cryptosystem Based on Algebraic Coding Theory. JPL DSN Progress Report **44** (05 1978)
14. Kachigar, G., Tillich, J.: Quantum information set decoding algorithms. In: Lange, T., Takagi, T. (eds.) Post-Quantum Cryptography, PQCrypto 2017, Utrecht, The Netherlands, June 26-28, 2017. LNCS, vol. 10346, pp. 69–89. Springer (2017), https://doi.org/10.1007/978-3-319-59879-6_5
15. Kirshanova, E.: Improved quantum information set decoding. In: Lange, T., Steinwandt, R. (eds.) Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018, Fort Lauderdale, FL, USA, April 9-11, 2018, Proceedings. Lecture Notes in Computer Science, vol. 10786, pp. 507–527. Springer (2018), https://doi.org/10.1007/978-3-319-79063-3_24

16. Lee, P.J., Brickell, E.F.: An Observation on the Security of McEliece's Public-Key Cryptosystem. In: Günther, C.G. (ed.) Advances in Cryptology - EUROCRYPT '88, Davos, Switzerland, May 25-27, 1988. LNCS, vol. 330, pp. 275–280. Springer (1988), https://doi.org/10.1007/3-540-45961-8_25
17. Melchor, C.A., Aragon, N., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Persichetti, E., Zémor, G., Bourges, I.: Hamming quasi-cyclic (hqc). NIST PQC Round **2**, 4–13 (2018), https://pqc-hqc.org/documentation.html
18. Mukherjee, C.S., Maitra, S., Gaurav, V., Roy, D.: Preparing dicke states on a quantum computer. IEEE Transactions on Quantum Engineering **1**(1), 1–17 (2020). https://doi.org/10.1109/TQE.2020.3041479
19. National Institute of Standards and Technology: Post-Quantum Cryptography Standardization process. https://nist.gov/pqcrypto (2017)
20. Niederreiter, H.: A Public-key Cryptosystem Based on Shift Register Sequences. In: Proc. EUROCRYPT '85. pp. 35–39. Springer-Verlag, Berlin, Heidelberg (1986), http://dl.acm.org/citation.cfm?id=20110.20114
21. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information: 10th Anniversary Ed. Cambridge University Press (2010). https://doi.org/10.1017/CBO9780511976667
22. Perriello, S., Barenghi, A., Pelosi, G.: A Complete Quantum Circuit to Solve the Information Set Decoding Problem. In: Proc. of the IEEE International Conference on Quantum Computing and Engineering, QCE 2021, Broomfield, CO, USA, October 18-22, 2021 (Fully virtual event). IEEE (2021)
23. Prange, E.: The use of information sets in decoding cyclic codes. IRE Trans. Information Theory **8**(5), 5–9 (1962), https://doi.org/10.1109/TIT.1962.1057777
24. Roetteler, M., Naehrig, M., Svore, K.M., Lauter, K.E.: Quantum Resource Estimates for Computing Elliptic Curve Discrete Logarithms. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 241–270. Springer (2017), https://doi.org/10.1007/978-3-319-70697-9_9
25. Shor, P.W.: Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. SIAM Review **41**(2), 303–332 (1999), https://doi.org/10.1137/S0036144598347011