Quantum Sieving for Decoding Linear Codes: A Reading Project

Omar Abul-Hassan[*]
Department of Mathematics
Stanford University
Stanford, CA, USA

Abstract

The development of quantum computing presents significant security challenges for existing cryptographic systems. Among promising post-quantum cryptosystems, code-based cryptography—such as the McEliece cryptosystem—relies fundamentally on the difficulty of decoding random linear codes, a problem known as Syndrome Decoding (SDP). While classical decoding algorithms, notably Information Set Decoding (ISD), have been extensively studied, quantum algorithms pose new complexity threats through techniques like Grover's algorithm and quantum walks. In this project, I looked through quantum sieving methods, which leverage quantum nearest-neighbor search and quantum walks to systematically exploit the structure of the decoding search space. I included step-by-step explanations of quantum sieving, analyze complexity improvements over classical and simpler quantum methods, and discuss their direct implications for cryptographic security. I also talk about practical implementation challenges, such as quantum resource constraints and error correction, and identify important future research directions.

1 Introduction

The advent of quantum computing has introduced profound implications for modern cryptographic systems. Traditional cryptographic algorithms, particularly RSA and elliptic-curve cryptography (ECC), rely on computational problems that, while computationally challenging for classical computers, can be efficiently solved by quantum algorithms. Shor's groundbreaking algorithm [20] demonstrates that quantum computers can factor large integers and compute discrete logarithms exponentially faster than classical algorithms, effectively undermining the security foundations of these widely used cryptosystems. This emerging vulnerability has spurred extensive research into quantum-resistant cryptographic schemes, notably leading to the National Institute of Standards and Technology's (NIST) initiative for post-quantum cryptographic (PQC) standardization [21].

Among various PQC candidates, code-based cryptography, exemplified by the McEliece cryptosystem [15], stands out due to its reliance on the Syndrome Decoding Problem (SDP), a well-established NP-hard problem. The security of such schemes hinges on the computational hardness of decoding random linear error-correcting codes, making them a prime candidate for resisting quantum attacks. Nevertheless, the introduction of quantum computing technologies necessitates a rigorous reassessment of their underlying security assumptions.

Recent advances in quantum algorithms, specifically quantum-enhanced search methods such as Grover's algorithm [9] and quantum walks [2], pose new threats to code-based cryptographic schemes. These quantum methods significantly accelerate structured search tasks fundamental to decoding problems, leading to notable complexity reductions compared to classical methods. Quantum sieving,

[*]omarah@stanford.edu

originally developed within the context of lattice cryptography [1], has emerged as a particularly compelling approach for further exploiting structural properties of the decoding search space. By adapting lattice sieving techniques to linear codes, quantum sieving algorithms achieve complexity reductions beyond classical and simpler quantum methods [12].

This paper aims to provide a comprehensive review of quantum sieving methods applied to decoding linear codes. We begin by reviewing foundational concepts in linear coding theory and establishing the mathematical formalism behind the Syndrome Decoding Problem. We then systematically examine classical decoding algorithms, notably Information Set Decoding (ISD), detailing their historical developments and complexity improvements. Subsequently, we delve into foundational quantum computing concepts, carefully presenting Grover's search algorithm and quantum walks as essential building blocks.

The central focus of our review lies in quantum sieving—its historical roots, adaptation to linear codes, intuitive explanations, and detailed algorithmic constructions. We present an in-depth complexity analysis, highlighting the significant theoretical improvements achieved through quantum sieving compared to classical counterparts. Additionally, we discuss practical considerations such as quantum resource requirements, hardware limitations, and quantum error correction, providing context for the current feasibility of these methods.

By thoroughly exploring these aspects, this paper contributes to the understanding of quantum computational impacts on code-based cryptography and clarifies the necessary steps to maintain cryptographic security in a rapidly evolving quantum landscape. Finally, we outline potential future research directions, emphasizing both theoretical exploration and practical quantum hardware considerations.

2 Classical Decoding Methods

Classical decoding algorithms for linear codes, particularly random linear codes, form the bedrock upon which the security of code-based cryptography rests. The most important family of these algorithms is known as Information Set Decoding (ISD). We will first describe the general ISD framework and then delve into specific algorithms, starting with Prange's foundational method and progressing through increasingly sophisticated improvements.

2.1 Formal Definition of the Syndrome Decoding Problem

Before delving into the details of ISD, it's crucial to formally define the Syndrome Decoding Problem (SDP), which is the computational problem underlying the security of code-based cryptography.

Theorem 2.1 *Problem: Syndrome Decoding Problem (SDP)*

Given:

- A parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$ of a binary linear code C with length n and dimension k.
- A syndrome vector $s \in \mathbb{F}_2^{n-k}$.
- An integer t (the target error weight).

Find:

- An error vector $e \in \mathbb{F}_2^n$ such that:
    1. $He^T = s$ (The error vector matches the given syndrome)
    2. $\text{wt}(e) \le t$ (The Hamming weight of the error vector is at most t)

Explanation and Significance:

The SDP captures the essence of the decoding task. A codeword c is transmitted, and an error e occurs during transmission, resulting in a received word $y = c+e$. The syndrome s is calculated as $s = Hy^T$. Since $Hc^T = 0$ for any valid codeword c, we have $s = Hy^T = H(c+e)^T = Hc^T + He^T = He^T$. Thus, the syndrome depends only on the error vector e.

The SDP asks us to find an error vector e that is consistent with the observed syndrome s and has a weight at most t. The weight constraint reflects the assumption that errors are relatively rare. In many practical scenarios, and in most cryptographic applications, we are interested in the unique decoding setting, where the error weight t is small enough (less than half the minimum distance of the code) to guarantee that there is at most one solution to the SDP.

NP-Hardness: The crucial property of the SDP, which makes it suitable for cryptography, is its computational hardness. The SDP for general linear codes is known to be NP-hard [4]. This means that there is no known polynomial-time algorithm that can solve the SDP for all instances. Furthermore, even for random linear codes (which are the codes typically used in code-based cryptography), no efficient algorithm is known, and the best-known algorithms (both classical and quantum) have exponential time complexity. This hardness is the foundation of code-based cryptosystems like McEliece.

2.2 Information Set Decoding (ISD): The General Framework

The core problem, as established earlier, is the Syndrome Decoding Problem (SDP): given a parity-check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, a syndrome $s \in \mathbb{F}_2^{n-k}$, and a weight bound t, find an error vector $e \in \mathbb{F}_2^n$ such that $He^T = s$ and wt$(e) \leq t$. ISD algorithms approach this problem by making a crucial observation:

> *If we could identify a set of k linearly independent columns of H (an "information set") that correspond to error-free positions in the received word, the decoding problem would reduce to solving a system of linear equations.*

Let $I \subset \{1, 2, \ldots, n\}$ be a set of k indices. We denote by H_I the $(n-k) \times k$ submatrix of H consisting of the columns indexed by I. Similarly, e_I represents the vector formed by the components of e at the indices in I. The complement of I, denoted $\bar{I}$, contains the remaining $n-k$ indices, and $H_{\bar{I}}$, $e_{\bar{I}}$ are defined analogously.

If I is an information set, then H_I has full column rank (and for a random code, is almost certainly invertible). If, furthermore, we assume that all errors are located outside of I, then $e_I = 0$. The syndrome equation $He^T = s$ can be rewritten as:

$$H_I e_I^T + H_{\bar{I}} e_{\bar{I}}^T = s$$

Since we're assuming $e_I = 0$, this simplifies to:

$$H_{\bar{I}} e_{\bar{I}}^T = s$$

This is now a system of $n-k$ equations in $n-k$ unknowns (the components of $e_{\bar{I}}$). If $H_{\bar{I}}$ is invertible (which is highly likely if I is a true information set and H is random), we can solve this system using standard linear algebra techniques (e.g., Gaussian elimination) to find $e_{\bar{I}}$. We then reconstruct the full error vector e by setting $e_I = 0$ and $e_{\bar{I}}$ to the solution we just found. Finally, we check if wt$(e) \leq t$. If so, we have found a valid solution.

The fundamental challenge, of course, is that we do not know which positions are error-free. ISD algorithms deal with this by guessing information sets and hoping that the guess is correct. The basic structure of all ISD algorithms is therefore:

1. **Choose an information set I:** Select a set of k column indices of H.
2. **Solve the linear system:** Assuming $e_I = 0$, solve $H_{\bar{I}} e_{\bar{I}}^T = s$ for $e_{\bar{I}}$.
3. **Check the weight:** Construct $e = (0, e_{\bar{I}})$ and check if wt$(e) \leq t$. If so, return e. Otherwise, go back to step 1.

The different ISD algorithms (Prange, Stern, Dumer, BJMM, etc.) differ primarily in how they choose the information set I and how they solve (or partially solve) the resulting linear system.

2.3 Prange's Algorithm (1962)

Prange's algorithm [19] is the simplest ISD algorithm. It makes the most straightforward guess for the information set: it chooses k indices uniformly at random.

Algorithm (Prange):

1. **Random Information Set Selection:** Randomly choose a set I of k column indices of H.
2. **Gaussian Elimination:** Perform Gaussian elimination on H to put it in the form $[H_I|H_{\bar{I}}]$, where H_I is an $(n-k) \times k$ matrix. If H_I is not invertible, go back to step 1. If we multiply on the left with an invertible S, we transform H into a systematic form $[I_{n-k}|\tilde{H}_{\bar{I}}]$.
3. **Solve Linear System:** Assuming $e_I = 0$, solve the system $\tilde{H}_{\bar{I}} e_{\bar{I}}^T = Ss$ for $e_{\bar{I}}$. (Note: Because of the systematic form this simply becomes $e_{\bar{I}}^T = Ss$).
4. **Weight Check:** Construct $e = (0, e_{\bar{I}})$ and check if $\text{wt}(e) \leq t$. If so, return e. Otherwise, go back to step 1.

Complexity Analysis of Prange's Algorithm: The running time of Prange's algorithm is dominated by the expected number of iterations of the loop (steps 1-4). Each iteration involves: Choosing a random set I (negligible cost), Gaussian elimination (cost $O(n^3)$, but can be made smaller), solving a linear system (already done in the Gaussian Elimination), and a weight check (cost $O(n)$).

The Gaussian elimination is the most expensive part, but it's still polynomial in n. The exponential cost comes from the probability of success in a single iteration. An iteration succeeds if and only if the randomly chosen set I contains no error positions.

The probability of choosing an error-free information set is:

$$P(\text{success}) = \frac{\binom{n-t}{k}}{\binom{n}{k}}$$

This is because there are $\binom{n}{k}$ total ways to choose k positions, and $\binom{n-t}{k}$ ways to choose k positions from the $n-t$ error-free positions.

The expected number of iterations is the reciprocal of this probability:

$$\text{Number of iterations} = \frac{1}{P(\text{success})} = \frac{\binom{n}{k}}{\binom{n-t}{k}}$$

Using Stirling's approximation for binomial coefficients ($n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$), and expressing everything in terms of the code rate $R = k/n$ and the relative error weight $\tau = t/n$, the complexity can be approximated as:

$$\text{Complexity} \approx 2^{cn}$$

where $c \approx H_2(\tau) - (1-R)H_2\left(\frac{\tau}{1-R}\right)$, and $H_2(x) = -x\log_2(x) - (1-x)\log_2(1-x)$ is the binary entropy function. For typical cryptographic parameters, $c \approx 0.1207$. So, the overall complexity is approximately $O(2^{0.121n})$.

2.4 Improvements: Stern, Dumer, and BJMM

The basic idea of Prange's algorithm is to find a set of k error-free positions. All subsequent improvements can be seen as more sophisticated ways of finding such sets, or, equivalently, of finding sets of positions that contain few errors.

2.4.1 Stern's Algorithm (1989)

Stern's algorithm [22] introduces the crucial concept of a meet-in-the-middle approach. The key idea is:

- **Split the error vector** Instead of assuming all k positions in the information set I are error-free, we allow a small number of errors, p, within I. We split I into two halves, I_1 and I_2, each of size $k/2$ (assume for simplicity that k is even). We also split the error vector e into three parts: $e = (e_{I_1}, e_{I_2}, e_{\bar{I}})$, where $\bar{I}$ is the complement of I.
- **Assume few errors in each half**: We assume that the number of errors in I_1 is $p/2$ and the number of errors in I_2 is also $p/2$ (again, assume p is even).
- We now create two lists, L_1: Contains all possible values of $H_{I_1}e_{I_1}^T$ where $\text{wt}(e_{I_1}) = p/2$ and L_2: Contains all possible values of $s - H_{I_2}e_{I_2}^T$ where $\text{wt}(e_{I_2}) = p/2$.

Collision search: We search for a collision between the two lists. A collision occurs when an element in L_1 is equal to an element in L_2. That is, we look for e_{I_1} and e_{I_2} such that:

$$H_{I_1}e_{I_1}^T = s - H_{I_2}e_{I_2}^T$$

This is equivalent to $H_{I_1}e_{I_1}^T + H_{I_2}e_{I_2}^T = s$, or $H_I e_I^T = s$. If we find such a collision, we can then solve for $e_{\bar{I}}$ as before, assuming there are $t - p$ errors in the remaining positions.

The key advantage of Stern's algorithm is that searching for a collision between two lists of size L is much faster than searching a single list of size L^2. The complexity is roughly $O(L)$ (using efficient sorting or hashing techniques), while the probability of success increases. The optimal choice of p is chosen to balance the list sizes and the probability of finding a collision. Stern's algorithm achieves a complexity of approximately $O(2^{0.117n})$.

2.4.2 Dumer's Algorithm (1991)

Dumer's algorithm [8] generalizes Stern's approach by recursively splitting the information set and the error vector into more and more parts. Instead of just two lists, Dumer's algorithm might use four lists, eight lists, and so on. This leads to a multi-level meet-in-the-middle search.

While the details of Dumer's algorithm are more complex, the core idea is the same: exploit the birthday paradox by splitting the problem into smaller subproblems and searching for collisions between lists. The recursive splitting allows for a finer-grained search and further reduces the complexity exponent, although the improvement over Stern's algorithm is relatively modest.

2.4.3 BJMM Algorithm (2012)

The Becker-Joux-May-Meurer (BJMM) algorithm [3] is the most sophisticated classical ISD algorithm to date. It builds upon the ideas of Stern and Dumer but introduces several key innovations:

- **Multi-Level Meet-in-the-Middle**: BJMM uses a multi-level meet-in-the-middle approach, similar to Dumer's, but with a more carefully designed hierarchy of lists and collisions.
- **Representation Technique**: BJMM introduces the concept of "representations." Instead of searching for a single error vector e, it searches for multiple representations of e as a sum of vectors with specific weight distributions. This increases the probability of finding a collision in the meet-in-the-middle steps. For example, instead of searching for e directly, it might search for $e_1 + e_2 = e$, where e_1 and e_2 have specific weights on different parts of the codeword. There are exponentially many ways to represent e in this way, increasing the chances of finding a useful collision.
- **Nearest Neighbor Search (Later Improvements)**:The most modern versions of BJMM, and related ISD algorithms, incorporate techniques from nearest neighbor search. This is a crucial connection to quantum sieving. The idea is to find vectors that are "close" to each other in Hamming distance more efficiently than by brute-force comparison. These techniques are borrowed from lattice sieving algorithms, as we will see later. May and Ozerov (2015) [14] made significant improvements using this approach.

The combination of these techniques allows BJMM to achieve the lowest known classical complexity for decoding random linear codes, around $O(2^{0.102n})$ for full-distance decoding, and $O(2^{0.0494n})$ for half-distance decoding.

2.5 Classical Complexity Analysis: Recap

The complexities listed are asymptoticcomplexities, meaning they describe the behavior of the algorithms as the code length n goes to infinity. The hidden constants can be significant, and the practical performance of these algorithms depends on the specific code parameters and implementation details. It's crucial to understand that all classical ISD variants, from Prange to BJMM, still have exponential time complexity. The improvements lie in reducing the constant in the exponent, making the algorithms faster but not fundamentally changing their exponential nature. This exponential hardness is the core reason code-based cryptography is considered a strong candidate for post-quantum security.

3 Quantum Algorithms: Foundations

Classical decoding algorithms, even the most sophisticated ones like BJMM, fundamentally rely on cleverly searching through an exponentially large space of possible error vectors. Quantum computing offers the potential to speed up this search, although it does not make the decoding problem tractable in polynomial time. This section introduces the key quantum algorithmic techniques that have been applied to the decoding problem: Grover's search algorithm and quantum walks.

3.1 Quantum Computing Overview

Before delving into specific algorithms, let's briefly review the core concepts of quantum computing that are relevant to our discussion. A more thorough treatment can be found in standard textbooks like [18].

- **Qubits and Superposition:** Unlike classical bits, which can be either 0 or 1, a quantum bit (qubit) can exist in a superposition of both states simultaneously. The state of a qubit can be represented as a vector in a two-dimensional complex vector space:

  $$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

  where α and β are complex numbers (amplitudes) such that $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ represents the probability of measuring the qubit as 0, and $|\beta|^2$ represents the probability of measuring it as 1. A system of n qubits can exist in a superposition of all 2^n possible classical states.
- **Unitary Transformations:** Quantum computations are performed by applying unitary transformations to the qubits. A unitary transformation is a linear operation that preserves the norm of the quantum state (i.e., it keeps the total probability equal to 1). Unitary transformations are reversible, which is a key difference from many classical computations. Mathematically, a unitary transformation is represented by a unitary matrix U (meaning $U^\dagger U = I$, where $U^\dagger$ is the conjugate transpose of U).
- **Measurement:** When we measure a qubit in a superposition state, we obtain a classical outcome (either 0 or 1) with a probability determined by the squared magnitudes of the amplitudes. The measurement collapses the superposition to the measured state. For example, if we measure the qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, we obtain 0 with probability $|\alpha|^2$ and 1 with probability $|\beta|^2$, and the qubit's state becomes $|0\rangle$ or $|1\rangle$ respectively.
- **Quantum Interference and Amplitude Amplification:** Quantum algorithms often exploit the phenomenon of interference. The amplitudes α and β are complex numbers, and they can interfere constructively or destructively. By carefully designing the unitary transformations, we can make the amplitudes of "good" states (e.g., solutions to a search problem) interfere constructively, increasing their probability of being measured, while the amplitudes of "bad" states interfere destructively, decreasing their probability. This is the core idea behind amplitude amplification, a generalization of Grover's search.

3.2 Grover's Search Algorithm

Grover's algorithm [9] is a fundamental quantum algorithm for unstructured search. Suppose we have a function $f : \{0,1\}^n \to \{0,1\}$, and we want to find an input x such that $f(x) = 1$. We are assuming that we have no prior knowledge about the structure of f – it's a "black box" function.

Classically, if we have no information about the structure of f, we might have to evaluate f on all 2^n possible inputs in the worst case. Grover's algorithm can find a solution (if one exists) using only $O(\sqrt{2^n})$ evaluations of f.

Algorithm (Grover):

1. **Initialization:** Start with a uniform superposition of all possible inputs:

$$|\psi_0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x\rangle = H^{\otimes n}|0\rangle^{\otimes n}$$

   This can be created by applying the Hadamard transform (H) to each qubit of an initial state $|00\ldots0\rangle$. The Hadamard transform on a single qubit is defined as:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

2. **Grover Iteration:** Repeat the following two steps R times (where R is a carefully chosen number, approximately $\frac{\pi}{4}\sqrt{2^n}$ if there's a unique solution):

   (a) **Oracle Reflection (O_f):** Apply a unitary transformation O_f that flips the sign of the amplitude of the solution state(s). If $f(x) = 1$, then $O_f|x\rangle = -|x\rangle$; otherwise, $O_f|x\rangle = |x\rangle$. This can be implemented with a quantum circuit that computes $f(x)$ and uses the result to control a phase shift. More formally:

$$O_f = I - 2|x_s\rangle\langle x_s|$$

   where $|x_s\rangle$ is the solution state(s) we're trying to find.

   (b) **Diffusion Transform (D):** Apply a unitary transformation D called the "diffusion operator" or "inversion about the mean" operator. This operator can be expressed as:

$$D = 2|\psi_0\rangle\langle\psi_0| - I$$

   where $|\psi_0\rangle$ is the initial uniform superposition state and I is the identity operator. This operation can be efficiently implemented using Hadamard gates and a conditional phase shift. The diffusion transform amplifies the amplitude of the solution state(s) and diminishes the amplitudes of the non-solution states.

3. **Measurement:** Measure the final state in the computational basis. With high probability, the measurement outcome will be a solution x such that $f(x) = 1$.

Grover's Algorithm Applied to ISD:

In the context of Prange's ISD algorithm, we can use Grover's algorithm to speed up the search for an error-free information set. The function $f(x)$ in this case would be a function that takes a set of k indices (represented as a binary string x) and checks if that set is an error-free information set. This check involves performing Gaussian elimination and verifying the weight of the resulting error vector, as described in Prange's algorithm. Since checking a given information set takes polynomial time, the quantum oracle can be implemented efficiently.

Bernstein (2010) [5] showed that applying Grover's algorithm to Prange's ISD reduces the complexity exponent by approximately half.

Theorem 3.1 *Lemma (Grover's Speedup for ISD, Bernstein 2010) [5]* Applying Grover's search to Prange's ISD algorithm results in a quadratic speedup. The quantum complexity is approximately $O(2^{0.06035n})$, compared to the classical complexity of $O(2^{0.121n})$.

3.3 Quantum Walks and Structured Searching

While Grover's algorithm provides a significant speedup for unstructured search, it doesn't exploit any potential structure in the search space. For many problems, including decoding, the search space does have structure, and quantum walks can leverage this structure to achieve even better performance.

Classical Random Walks: A classical random walk on a graph is a process where a "walker" starts at a vertex and, at each step, moves to a randomly chosen neighboring vertex. Random walks are used in many classical algorithms, including algorithms for searching, sampling, and optimization.

Quantum Walks: Quantum walks are the quantum analog of classical random walks. Instead of a probability distribution over vertices, a quantum walk involves a superposition of states corresponding to the vertices of the graph. The evolution of the quantum walk is governed by a unitary operator that mixes the amplitudes of neighboring vertices.

There are two main types of quantum walks:

- **Discrete-Time Quantum Walks:** These walks proceed in discrete time steps, similar to classical random walks. A "coin" operator is often used to control the direction of the walk. The state of the walker is typically represented by a superposition over pairs (v, c), where v is a vertex and c is a "coin state" that determines the direction of the next step. The evolution consists of alternating coin and shift operations.
- **Continuous-Time Quantum Walks:** These walks evolve continuously in time, governed by a Hamiltonian operator related to the adjacency matrix of the graph.

For our purposes, discrete-time coined quantum walks are most relevant.

Key Advantages of Quantum Walks:

- **Interference:** Unlike classical random walks, quantum walks exhibit interference between different paths. This interference can be constructive (increasing the probability of reaching the target state) or destructive (decreasing the probability of staying in non-target states).
- **Faster Hitting Time:** For many graphs, the hitting time (the expected time to reach a target vertex) of a quantum walk is significantly smaller than that of a classical random walk.
- **Exploiting Structure:** Quantum walks can be designed to exploit the specific structure of the search space. This is crucial for improving the performance of ISD algorithms.

Quantum Walks for ISD:

Kachigar and Tillich (2017) [11] showed how to apply quantum walk techniques to improve the complexity of ISD algorithms. Their approach involves:

1. **Constructing a Search Graph**: Define a graph where the vertices represent partial solutions to the decoding problem (e.g., partial error vectors that satisfy some of the parity-check equations). Edges connect vertices that can be combined to form "larger" partial solutions, getting closer to a full solution. The graph structure reflects the way partial solutions can be combined. This is often a Johnson graph or a variant of it. A Johnson graph $J(n, r)$ has vertices that represent all r-element subsets of an n-element set, and two vertices are adjacent if their subsets differ in exactly one element.
2. **Defining a Quantum Walk:** Define a unitary operator that governs the evolution of a quantum walk on this graph. This operator typically involves a "coin flip" operation (which determines the direction of the walk) and a "shift" operation (which moves the walker to a neighboring vertex). The coin flip and shift operations are carefully designed to exploit the structure of the graph and promote constructive interference towards solutions.
3. **Using Amplitude Amplification**: Use amplitude amplification, a generalization of Grover's search, to boost the probability of finding a "marked" vertex, which corresponds to a complete solution to the decoding problem. Amplitude amplification repeatedly applies the quantum walk operator and a reflection operator that inverts the phase of the marked vertices.

Theorem 3.2 (Quantum ISD Complexity, Kachigar–Tillich, 2017 [11]) *Quantum Information-Set Decoding algorithms, leveraging quantum walks, achieve a time complexity of approximately:*

$$T_{\text{QW-ISD}} \approx 2^{0.05869n},$$

for decoding random linear codes of length n. This complexity surpasses both Prange's original algorithm with Grover's speedup and classical ISD techniques by explicitly taking into account the structure of the SDP. This represents a concrete improvement over the Grover-based approach.

The specific quantum walk algorithms used by Kachigar and Tillich build upon earlier work on quantum algorithms for the subset-sum problem, which has a similar structure to the decoding problem.

Why Quantum Walks are Better than Grover for ISD:

Grover's algorithm provides a quadratic speedup for unstructured search. However, the decoding problem (and many other problems in cryptography) have structure. Quantum walks can be designed to exploit this structure, leading to better-than-quadratic speedups. In the case of ISD, the structure comes from the fact that we can build up solutions by combining partial solutions that satisfy subsets of the parity-check equations. The quantum walk explores the space of partial solutions in a way that favors combinations that are more likely to lead to a full solution.

4 Quantum Sieving

Having established the foundations of classical decoding algorithms and basic quantum algorithmic techniques, we now turn to the core topic of this paper: quantum sieving. Quantum sieving represents a significant advance in quantum algorithms for decoding linear codes. It is not a completely new algorithm but rather a powerful combination of ideas:

- **Classical Sieving:** A technique borrowed and adapted from lattice-based cryptography.
- **Quantum Nearest-Neighbor Search:** Using quantum algorithms to accelerate the core operation within sieving.
- **Quantum Walks:** Structuring the search for close vectors using quantum walks.

The result is an algorithm that, while still exponential in complexity, achieves the best-known quantum time complexity for decoding random linear codes.

4.1 Historical Origins (Lattice Sieving)

To understand quantum sieving for codes, it is helpful to first understand its origins in lattice-based cryptography. Lattice problems, like the Shortest Vector Problem (SVP) and the Closest Vector Problem (CVP), are believed to be hard even for quantum computers, making them another foundation for post-quantum cryptography.

Classical lattice sieving algorithms, introduced by Ajtai, Kumar, and Sivakumar (AKS) in 2001 [1], provide a way to solve SVP (and related problems) that, while still exponential, is faster than brute-force search. The core idea is:

1. **Generate a large list of lattice vectors:** These vectors are typically much longer than the shortest vector in the lattice.
2. **Sieving:** Iteratively combine pairs of vectors in the list to produce shorter vectors. This process is called "sieving" because it filters out the long vectors and keeps the short ones. The combination is usually a simple vector addition or subtraction.
3. **Repeat:** Repeat the sieving process until the list contains vectors that are close to the shortest vector.

The key insight is that if you have two relatively short lattice vectors, their sum or difference is likely to be even shorter, provided the vectors are somewhat close to each other. The sieving step, therefore, is essentially a nearest-neighbor search in the high-dimensional space of the lattice. Classical sieving algorithms use sophisticated data structures and techniques (like locality-sensitive hashing) to perform this nearest-neighbor search as efficiently as possible [17, 16].

Quantum sieving algorithms for lattices, such as those in [13, 6, 5], build upon these classical sieving methods but use quantum algorithms (Grover's search, quantum walks) to speed up the nearest-neighbor search. This leads to a provable quantum speedup, although the algorithms remain exponential.

4.2 Adapting Sieving to Linear Codes

The crucial step is to adapt the sieving concept from the continuous space of lattices ($\mathbb{R}^n$) to the discrete space of binary vectors ($\mathbb{F}_2^n$). This adaptation was first proposed in [10] and further developed in [7]. The key analogies are:

Concept	Lattice Sieving	Code Sieving
Search Space	High-dimensional lattice	Binary vectors ($\mathbb{F}_2^n$)
"Short" Vector	Vector with small Euclidean norm	Vector with small Hamming weight
Combination Operation	Vector addition/subtraction	Bitwise XOR (addition in $\mathbb{F}_2^n$)
"Closeness"	Small Euclidean distance	Small Hamming distance
Nearest-Neighbor Search	Find vectors close in Euclidean distance	Find vectors with small Hamming distance
"Sieving"	Iteratively combine to get shorter vectors	Iteratively combine to satisfy more constraints

Instead of looking for short vectors in a lattice, we're looking for low-weight error vectors that satisfy the syndrome equation $He^T = s$. The "sieving" process involves:

1. **Generating a List:** Start with a large list L of binary vectors of length n and weight w. These are candidate error vectors, but they don't necessarily satisfy the syndrome equation (i.e., they are not necessarily codewords + the target error).
2. **Imposing Constraints:** Iteratively combine pairs of vectors in the list to produce new vectors that satisfy more parity-check equations. We'll explain this in detail shortly.
3. **Reaching the Solution:** After enough iterations, we hope to obtain a vector that satisfies all the parity-check equations (i.e., it's a codeword) and has the correct weight w. This is our solution to the decoding problem.

4.3 Motivation and Intuition: Why Sieving Works

The core idea behind sieving (both classical and quantum) is to systematically explore the search space, rather than randomly guessing solutions (like in basic ISD).

- **Classical ISD (Prange):** Imagine trying to find a specific needle in a haystack. Prange's algorithm is like randomly picking handfuls of hay, hoping to find the needle. Most of the time, you find nothing but hay.
- **Sieving (Intuitive Picture):** Imagine the haystack is made of magnetic hay, and the needle is also magnetic. Sieving is like taking two handfuls of hay, putting them close together, and seeing if any pieces stick together. If they do, you've likely gotten closer to the needle (or you might even have found it!). You keep repeating this process, combining "clumps" of hay that are more likely to contain the needle.

Classical sieving maintains a large collection of "candidate" error vectors. It then combines pairs of these candidates, hoping that the combination will be "closer" to a true solution (i.e., satisfy more parity-check equations). However, classical sieving has to perform these comparisons individually which makes the process slow. **Quantum Advantage:** Quantum sieving uses the same basic idea, but it leverages quantum algorithms to perform the crucial "combining" step much more efficiently. This is done using the quantum nearest-neighbor search. Instead of checking pairs of candidates one by one, a quantum computer can examine many pairs simultaneously in superposition.

4.4 Quantum Sieving: Detailed Construction

Quantum sieving conssits of three steps:

Step 1: Structured List Generation

1. **Create an Exponentially Large List:** Begin by generating a large list, denoted as L, of candidate error vectors. These vectors are binary strings of length n with a Hamming weight of w, where w is the target error weight for the decoding problem. Importantly, these vectors

are not necessarily solutions to the syndrome decoding problem ($He^T = s$); they are simply vectors with the correct weight. We can think of them as "partial solutions" or "building blocks."

2. **List Size ($|L|$):** The size of this list is a crucial parameter. It needs to be large enough so that we have a reasonable chance of finding pairs of vectors that can be combined to form a solution (we'll discuss this "combining" in the next step). However, the list shouldn't be too large, or it will become computationally expensive to manage. A typical choice is $|L| \approx 2^{\lambda n}$, where λ is a constant that is optimized to minimize the overall algorithm's complexity.
3. **Initial Vectors:** Initially, the list L is populated with vectors drawn uniformly at random from the set of all binary vectors of length n and weight w. We denote this set as S_w^n.
4. **Tower of Codes:** The sieving process will iteratively impose constraints on the vectors in the list. To do this, we define a sequence of nested linear codes:

$$\mathbb{F}_2^n = C_0 \supset C_1 \supset C_2 \supset \cdots \supset C_{n-k} = C$$

   where C is the original $[n, k]$ linear code we want to decode. Each C_i is a subcode of C_{i-1}, and the dimension of C_i is $n - i$. Each C_i is defined by a set of i parity-check equations (i.e., the first i rows of the parity-check matrix H). The idea is that as we move down the tower of codes, we are enforcing more and more constraints, getting closer and closer to the actual code C. We can think of these C_i codes as a series of increasingly restrictive "sieves."
5. **Goal of Sieving:** The sieving steps aim to generate a sequence of lists $L_0, L_1, ..., L_{n-k}$. The intial list, L_0 is the initial list we created ($L_0 = L$). The list L_i will contain vectors that are not only of weight w, but are also codewords of the code C_i.

Step 2: Quantum Nearest-Neighbor Search

This is the heart of the quantum sieving algorithm and where the quantum speedup comes from. In each iteration of the sieving process, we need to find pairs of vectors in the current list L_{i-1} that, when combined, satisfy an additional parity-check equation. More precisely, suppose we have a list L_{i-1} of vectors that are codewords of C_{i-1} (i.e., they satisfy the first $i - 1$ parity-check equations). We want to find pairs $(x, y) \in L_{i-1} \times L_{i-1}$ such that:

1. $x + y$ has weight w (where "+" denotes bitwise XOR, which is addition in $\mathbb{F}_2^n$).
2. $x + y$ satisfies the i-th parity-check equation.

This can be viewed as a nearest-neighbor search problem in Hamming space. We are looking for pairs of vectors (x, y) that are "close" to each other in the sense that their sum has a specific weight and satisfies a constraint (the i-th parity check).

- **Classical Approach:** Classically, finding such pairs would involve comparing many pairs of vectors in L_{i-1}. If the list has size N, a naive approach would require $O(N^2)$ comparisons. More sophisticated classical sieving algorithms use techniques like locality-sensitive hashing (LSH) to reduce this cost, but it remains a significant bottleneck.
- **Quantum Approach:** Quantum sieving uses a quantum nearest-neighbor search algorithm to find these pairs much more efficiently. This is the core of the quantum speedup. There are several ways to implement the quantum nearest-neighbor search, including:
  - **Grover's Algorithm:** A straightforward approach is to use Grover's algorithm to search for pairs (x, y) that satisfy the desired conditions. This provides a quadratic speedup over classical search.
  - **Quantum Walks:** More sophisticated approaches use quantum walks on a graph where vertices represent vectors in L_{i-1}, and edges connect vectors that are "close" in some sense (e.g., their sum has low weight or satisfies many parity checks). Quantum walks, combined with amplitude amplification (a generalization of Grover's search), can find these close pairs with a complexity that is better than both classical search and Grover's algorithm alone.
  - **Locality Sensitive Filtering (LSF):** A crucial technique used in both classical and quantum sieving is locality-sensitive filtering (LSF). The idea is to group vectors into "buckets" based on some hash function, such that vectors that are close to each

other are likely to end up in the same bucket. This reduces the search space for the nearest-neighbor search. Quantum sieving algorithms often use LSF in conjunction with quantum walks. The quantum walk operates on a graph where vertices represent buckets, and edges connect buckets that are likely to contain close vectors.

- **Mathematical Formulation of Quantum Search:** The quantum algorithm operates on a superposition of all pairs of vectors in L_{i-1}. This allows it to "check" many pairs simultaneously. We can represent the initial state as:

$$\frac{1}{|L_{i-1}|} \sum_{x,y \in L_{i-1}} |x, y\rangle$$

  The unitary transformations within the quantum walk (or Grover's search) are designed to amplify the amplitudes of pairs (x, y) that satisfy the desired conditions (small Hamming distance and satisfying the parity-check equation) and diminish the amplitudes of other pairs.

Step 3: Collision Identification and Solution Reconstruction

After the quantum nearest-neighbor search, we have a quantum state that, with high probability, is a superposition of "good" pairs (x, y)—pairs whose sum has the correct weight and satisfies the i-th parity-check equation.

1. **Measurement and Combination:** We measure the quantum state. This collapses the superposition and gives us a classical pair (x, y). We then compute their sum (bitwise XOR): $z = x + y$.
2. **Classical Verification:** We classically check if z satisfies the i-th parity-check equation and if it has weight w. If it satisfies both conditions, we add z to the new list L_i.
3. **Iteration:** We repeat Steps 2 and 3 for each of the $n - k$ parity-check equations, obtaining lists $L_1, L_2, \ldots, L_{n-k}$.
4. **Final Solution:** The final list L_{n-k} should, with high probability, contain the solution to the decoding problem (i.e., the error vector e). Since L_{n-k} contains vectors that satisfy all the parity checks of the original code C, any vector in L_{n-k} with the correct weight w is a solution.

4.5 Quantum Sieving within the ISD Framework

The quantum sieving algorithm described above is not a standalone decoding algorithm. Instead, it is used as a subroutine within the broader Information Set Decoding (ISD) framework. Recall that the basic ISD approach involves:

- Guessing an Information Set: Choosing a set of k column indices, hoping they correspond to error-free positions.
- Solving a Linear System: Assuming the chosen set is error-free, solving a simplified linear system.
- Checking the Solution: Verifying if the resulting error vector has the correct weight.

Quantum sieving replaces the "solving a linear system" step. Instead of directly solving for the error vector on the complement of the information set, we use quantum sieving to find candidates for the error vector. The crucial change from classical to quantum sieving ISD is in the subroutine. Here's how it fits in:

- Choose an Augmented Information Set: Instead of just choosing a set I of size k, we choose a slightly larger set J of size $k + \ell$, where ℓ is a parameter. This set J is called an augmented information set. The hope is that J contains most of the error-free positions but may contain a few error positions (say, p errors).
- Form a Subproblem: We form a smaller, "easier" decoding problem by focusing on the columns of H indexed by J. Let H_J be the submatrix of H corresponding to these columns, and let s' be a part of the syndrome vector. We now have a new syndrome decoding problem: find a vector e' of length $k + \ell$ and weight p such that $H_J(e')^T = s'$.

- Use Quantum Sieving: We use the quantum sieving algorithm to find many solutions to this subproblem. That is, we generate a list of candidate error vectors e' that have weight p and satisfy the syndrome equation for the submatrix H_J.
- Extend and Check: For each candidate e' found by the sieving algorithm, we "extend" it to a full-length error vector e by assuming that the remaining positions (outside of J) have the remaining errors. We then check if this full-length vector e has the correct weight t and satisfies the original syndrome equation $He^T = s$. If so, we've found a solution. If not, we try the next candidate from the sieving algorithm or choose a new augmented information set J.

Key Idea: The quantum sieving algorithm is used as a black box to generate a list of candidate solutions to a smaller, structured decoding problem. The outer ISD framework then iterates through these candidates and checks if any of them lead to a solution of the original decoding problem.

Why this is better than basic Quantum ISD:

- By allowing a few errors (p errors) within the augmented information set J, we can make the subproblem easier to solve using quantum sieving than directly searching for a completely error-free information set.
- Quantum sieving gives a speedup within each iteration, and the outer loop benefits by checking more potential solutions generated by quantum sieving using quantum walks and superposition.

In summary, quantum sieving replaces the simple "solve a linear system" step in basic ISD with a much more sophisticated (and quantum-accelerated) search for candidate error vectors. The overall ISD framework remains the same, but the subroutine for finding candidate solutions is significantly enhanced.

5 Security Implications for Code-Based Cryptography

The advancements in quantum algorithms, particularly quantum sieving, have significant implications for the security of code-based cryptosystems. While these algorithms do not break the underlying NP-hardness of the decoding problem, they do reduce the computational complexity, effectively shrinking the security margin offered by these schemes. This necessitates a reevaluation of cryptographic parameters and a deeper understanding of the practical challenges in implementing quantum attacks.

5.1 Impact on McEliece and Related Schemes

The McEliece cryptosystem [15], and its variants, are among the leading candidates for post-quantum cryptography. Their security rests on the assumed hardness of decoding a random linear code (or a suitably disguised structured code). Classical ISD algorithms, like BJMM [3], provide the best-known attacks, with a complexity of roughly $O(2^{0.102n})$ for full-distance decoding and $O(2^{0.0494n})$ for half-distance decoding (where n is the code length). These classical complexities define the pre-quantum security margin.

Quantum sieving, as discussed, further reduces this complexity. The best known quantum sieving algorithms (and quantum ISD algorithms in general) achieve a complexity of approximately $O(2^{0.058n})$ (or slightly better with ongoing refinements). This is a substantial reduction in the exponent, though the problem remains exponential.

Specifically, the impact can be quantified as follows:

- Classical Security: If a McEliece variant was designed to offer 128 bits of classical security (meaning the best classical attack requires approximately 2^{128} operations), this was based on estimates using classical ISD algorithms.
- Quantum Security (with Grover): A naive application of Grover's algorithm would roughly halve the exponent, reducing the security to around 64 bits.
- Quantum Security (with Quantum Walks/Sieving): More sophisticated quantum algorithms, like quantum walks and quantum sieving, further reduce the effective security. The current

best estimate, using quantum sieving, reduces the effective security level to approximately 77 bits, if the code parameters remain unchanged. This is a significant reduction, though still far from being practically breakable.

5.2 Quantum vs. Classical Security Margins

The reduction in the complexity exponent directly translates to a shrinking security margin. If a cryptosystem aims for a security level of 2^λ operations (e.g., $\lambda = 128$ for 128-bit security), the parameters (code length, error correction capability) must be chosen so that the best-known attack requires at least 2^λ operations.

Quantum algorithms, by reducing the complexity exponent, force us to increase the code parameters to maintain the same security level. For example, if a classical algorithm has complexity $O(2^{cn})$ and a quantum algorithm has complexity $O(2^{c'n})$ with $c' < c$, then to maintain a security level of 2^λ, the code length n must be increased in the quantum setting. The ratio c/c' gives an indication of how much larger the parameters need to be.

Theorem 5.1 *Theorem (Quantum Security Margin)* To maintain equivalent security levels under quantum sieving attacks, code length parameters must, as an approximation, be increased according to the following ratio:

$$n_{\text{quantum}} \approx \frac{c_{\text{classical}}}{c_{\text{quantum}}} \cdot n_{\text{classical}}$$

Where $c_{\text{classical}}$ is the complexity exponent of the best classical decoding algorithm (like BJMM), and c_{quantum} is the exponent for the best quantum decoding algorithm (like quantum sieving). Using the values we have previously found, we can estimate:

$$n_{\text{quantum}} \approx \frac{0.096}{0.058} \cdot n_{\text{classical}} \approx 1.6 \cdot n_{\text{classical}}$$

This suggests an approximately 60% increase in code length is needed to counteract the quantum speedup.

5.3 Adjusting Cryptographic Parameters for Quantum Resistance

The practical implication of the reduced security margin is that code-based cryptosystems need to use larger parameters to maintain their security against quantum attacks. This typically means:

- Increasing the code length (n): This is the most direct way to increase the complexity of decoding.
- Adjusting the code rate ($R = k/n$): The code rate affects the complexity exponent. Lower rate codes (smaller k for a given n) are generally harder to decode.
- Choosing different code families: While quantum sieving applies to generic linear codes, some code families might be more or less resistant to specific quantum algorithms. Research into alternative code constructions is ongoing.

The NIST Post-Quantum Cryptography standardization process [21] takes these quantum speedups into account when evaluating and selecting candidate algorithms. The recommended parameters for code-based schemes in the NIST process are significantly larger than those used in pre-quantum cryptography to ensure an adequate security margin against both classical and quantum attacks.

6 Practical Challenges and Implementation Considerations

While quantum sieving algorithms offer significant theoretical speedups, it's crucial to remember that they are, at present, theoretical. Building and operating large-scale, fault-tolerant quantum computers is an enormous technological challenge. Several practical factors limit the immediate threat posed by quantum sieving:

6.1 Quantum Resource Requirements (Qubits, QRAM)

Quantum sieving algorithms, especially those based on quantum walks, have substantial resource requirements:

- **Qubits:** The algorithms require a large number of qubits to represent the superpositions involved in the quantum search and nearest-neighbor finding. The exact number depends on the specific algorithm and parameters, but it's typically in the thousands or even millions of logical qubits.
- **Quantum RAM (QRAM):** Many quantum algorithms, including quantum sieving, assume access to quantum RAM (QRAM). QRAM allows a quantum computer to access arbitrary memory locations in superposition. This is a very powerful capability, but building large-scale, reliable QRAM is a major technological hurdle. The size of the QRAM needed for quantum sieving is related to the size of the lists used in the sieving process, which can be exponential in the code length.
- **Circuit Depth and Coherence Time:** Quantum computations are limited by the coherence time of the qubits – the amount of time they can maintain a superposition before decoherence (errors) occurs. Quantum sieving algorithms, particularly those with multiple levels of quantum walks, can require deep quantum circuits (many sequential operations). The deeper the circuit, the longer the coherence time required.

6.2 Quantum Error Correction (QEC) Overhead

Current quantum computers are noisy. To perform reliable computations, we need to use quantum error correction (QEC). QEC involves encoding logical qubits (the qubits used in the algorithm) using multiple physical qubits (the actual qubits in the hardware). This introduces significant overhead. The number of physical qubits required per logical qubit (the "overhead factor") can be quite large (tens, hundreds, or even thousands, depending on the desired error rate and the QEC code used). This significantly increases the overall qubit requirements.

6.3 Current Quantum Hardware Limitations

Current quantum computers are far from being able to implement the large-scale, fault-tolerant computations required for quantum sieving attacks on cryptographically relevant code sizes.

- **Limited Qubit Count**: Existing quantum computers have, at most, a few hundred physical qubits, and far fewer logical qubits after accounting for error correction.
- **Short Coherence Times**: Qubit coherence times are still relatively short, limiting the depth of quantum circuits that can be executed reliably.
- **Noisy Operations**: Quantum gates are imperfect, and errors accumulate during computation.
- **QRAM Challenges**: Large-scale, reliable QRAM is still in the early stages of development.

These limitations mean that, while quantum sieving algorithms pose a theoretical threat, the practical threat is still far off. However, the rapid pace of development in quantum computing necessitates ongoing research and proactive adjustments to cryptographic parameters. It's a race between advances in quantum algorithms and advances in quantum hardware.

References

[1] Miklos Ajtai, Ravi Kumar, and D Sivakumar. “A sieve algorithm for the shortest lattice vector problem”. In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. 2001, pp. 601–610. DOI: 10.1145/380752.380857.

[2] Andris Ambainis. “Quantum walks and their algorithmic applications”. In: *International Journal of Quantum Information* 1.04 (2003), pp. 507–518. DOI: 10.1142/S0219749903000383.

[3] Anja Becker et al. “Decoding random binary linear codes in 2 n/20: how 1+ 1= 0 improves information set decoding”. In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2012, pp. 520–536. DOI: 10.1007/978-3-642-29011-4_29.

[4] Elwyn R Berlekamp, Robert J McEliece, and Henk CA Van Tilborg. “On the inherent intractability of certain coding problems”. In: *IEEE Transactions on Information Theory* 24.3 (1978), pp. 384–386. DOI: 10.1109/TIT.1978.1055873.

[5] Daniel J. Bernstein et al. “Reusing Q-Walks for Multiple Walks”. In: *EUROCRYPT*. 2023, pp. 422–453. DOI: 10.1007/978-3-031-31704-7_15.

[6] André Chailloux and Johanna Loyer. “Lattice Sieving via Quantum Random Walks”. In: *Advances in Cryptology – ASIACRYPT 2021*. Lecture Notes in Computer Science. Springer, 2021. DOI: 10.1007/978-3-030-92077-4_3.

[7] Léo Ducas et al. *Asymptotics and Improvements of Sieving for Codes*. Cryptology ePrint Archive, Paper 2024/113. 2024. URL: https://ia.cr/2024/113.

[8] Ilya Dumer. “On minimum distance decoding of linear codes”. In: *Proceedings of the Fifth Joint Soviet-Swedish International Workshop on Information Theory* (1991), pp. 50–52.

[9] Lov K Grover. “A fast quantum mechanical algorithm for database search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. ACM. 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[10] Qian Guo, Thomas Johansson, and Phong Q. Nguyen. *A New Sieving-Style Information-Set Decoding Algorithm*. Cryptology ePrint Archive, Paper 2023/1323. https://ia.cr/2023/1323. 2023.

[11] Ghazal Kachigar and Jean-Pierre Tillich. “Quantum information set decoding algorithms”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2017, pp. 69–89. DOI: 10.1007/978-3-319-59879-6_5.

[12] Elena Kirshanova. “Improved quantum information set decoding”. In: *International Workshop on Post-Quantum Cryptography*. Springer. 2018, pp. 507–527.

[13] Thijs Laarhoven. “Search problems in cryptography: from fingerprinting to lattice sieving”. In: *PhD Thesis, Eindhoven University of Technology* (2015). URL: https://research.tue.nl/en/publications/search-problems-in-cryptography-from-fingerprinting-to-lattice-si.

[14] Alexander May and Ilya Ozerov. “On computing nearest neighbors with applications to decoding of binary linear codes”. In: *Annual international conference on the theory and applications of cryptographic techniques*. Springer. 2015, pp. 203–228. DOI: 10.1007/978-3-662-46803-6_8.

[15] Robert J. McEliece. “A public-key cryptosystem based on algebraic coding theory”. In: *DSN Progress Report* 42-44 (1978), pp. 114–116.

[16] Daniele Micciancio and Panagiotis Voulgaris. “Faster exponential time algorithms for the shortest vector problem”. In: *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*. SIAM. 2010, pp. 1468–1480. DOI: 10.1137/1.9781611973075.47.

[17] Phong Q Nguyen and Thomas Vidick. “Sieve algorithms for the shortest vector problem are practical”. In: *Journal of Mathematical Cryptology*. Vol. 2. 2. 2008, pp. 181–207. DOI: 10.1515/JMC.2008.009.

[18] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010. ISBN: 9781107002173.

[19] Eugene Prange. “The use of information sets in decoding cyclic codes”. In: *IRE Transactions on Information Theory* 8.5 (1962), S5–S9. DOI: 10.1109/TIT.1962.1057777.

[20] Peter W Shor. “Algorithms for quantum computation: discrete logarithms and factoring”. In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134. DOI: 10.1109/SFCS.1994.365700.

[21] National Institute of Standards and Technology. *Post-Quantum Cryptography Standardization*. `https://csrc.nist.gov/projects/post-quantum-cryptography`. 2022.

[22] Jacques Stern. "A method for finding codewords of small weight". In: (1989), pp. 106–113. DOI: `10.1007/3-540-50922-8_36`.