Syndrome Decoding Estimator

Andre Esser and Emanuele Bellini

Cryptography Research Center
Technology Innovation Institute
Abu Dhabi, UAE
{andre.esser, emanuele.bellini}@tii.ae

Abstract. The selection of secure parameter sets requires an estimation of the attack cost to break the respective cryptographic scheme instantiated under these parameters. The current NIST standardization process for post-quantum schemes makes this an urgent task, especially considering the announcement to select final candidates by the end of 2021. For code-based schemes, recent estimates seemed to contradict the claimed security of most proposals, leading to a certain doubt about the correctness of those estimates. Furthermore, none of the available estimates includes most recent algorithmic improvements on decoding linear codes, which are based on information set decoding (ISD) in combination with nearest neighbor search. In this work we observe that *all* major ISD improvements are build on nearest neighbor search, explicitly or implicitly. This allows us to derive a framework from which we obtain *practical* variants of all relevant ISD algorithms including the most recent improvements. We derive formulas for the practical attack costs and make those online available in an easy to use estimator tool written in python and C. Eventually, we provide classical and quantum estimates for the bit security of all parameter sets of current code-based NIST proposals.

Keywords: ISD, syndrome decoding, nearest neighbor, estimator, code-based

1 Introduction

The current NIST standardization process for post quantum schemes is announced to finally select proposals to be standardized around the end of 2021. After this initial selection it is still a long procedure until the final standards for the chosen schemes will be obtained. One major challenge will be the selection of secure parameter sets for standardization, which match the respective security levels given by NIST. To determine and evaluate parameter sets a precise estimation of the attack cost of the best known attacks on the schemes or the corresponding primitives is necessary.

Before such estimates can be derived efficiently, the best practical attacks must be identified. Code based schemes usually rely on the hardness of the syndrome decoding problem, which given a random matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, a syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$ and an integer ω asks to find an error vector $\mathbf{e} \in \mathbb{F}_2^n$ with exactly ω coordinates equal to one that satisfies $H\mathbf{e} = \mathbf{s}$. The best known algorithms to solve this problem all belong to a class of algorithms known as information set

decoding (ISD), initially discovered by Prange in 1962 [27]. Since then there have been numerous works building on the same initial idea [4, 8, 11, 14, 21, 22, 29]. Usually, these works study the problem for $\omega = cn$, where c is constant. For this choice they improve the asymptotic runtime exponent. However, all code based NIST PQC submissions rely on sublinear error weight, i.e. $\omega = o(n)$. In this setting the advantage of improved algorithms of the ISD class over Prange's initial approach has been shown to asymptotically vanish, i.e., they only affect second order terms [30]. Since usually these improvements come along with a polynomial overhead, it is per se not clear which algorithms actually yield practical improvements.

Estimators for concrete hardness approximation of the syndrome decoding problem have previously been studied in [15, 26] and more recently in [3]. So far these works consider only a subset of the mentioned improvements, not including the most recent variants, which are usually based on nearest neighbor search techniques [8, 22]. The omission of these improvements in [3] might be due to the use of practically costly, but theoretically close to optimal routines to instantiate the nearest neighbor search in the original works of [8, 22]. The BIKE submission gives a similar reasoning for disregarding these works in the security analysis based on polynomial overhead [1]. Contrary, we show that by substituting the used routines by more practical nearest neighbor search techniques these variants yield complexity improvements with regard to the cryptographic setting. Furthermore we uncover relations between all significant algorithmic improvements of the ISD class. More precisely, we show that all major ISD improvements use nearest neighbor search techniques, explicitly or implicitly. Using this relation we derive an algorithmic framework, which allows us to obtain variants of all advanced ISD techniques, including the improvements made by May-Meurer-Thomae (MMT) [21], Becker-Joux-May-Meurer (BJMM) [4], May-Ozerov [22] and Both-May [8]. Finally the framework allows us to analyze the complexity of all algorithms in a unified and practical model giving a fair comparison and concrete hardness estimations.

Related work In [26] Peters gives a concrete analysis of Stern's algorithm for decoding codes over $\mathbb{F}_q$ including the case of $q = 2$. Peters focuses especially on optimized strategies for the initial Gaussian elimination part of ISD algorithms, adopting techniques introduced in [7, 9]. While including some of these improvements in our estimates, we refrain from exhaustively optimizing this step. This allows us to keep the analysis and formulas comprehensible. Also note that for more recent ISD algorithms the complexity of the Gaussian elimination procedure does not dominate.

In [15] the authors present a non-asymptotic analysis of the MMT and BJMM algorithm, providing estimates for some selected parameter sets. Unfortunately no source code is provided to easily obtain estimates for parameter sets of more recent schemes. Also the analysis omits some practical details, as for instance the necessity for balanced weight distributions in successful runs of the algorithms. Also a heuristic approach to determine the number of iterations of the algorithms is used, whereas we use an exact computation.

The most recent study of concrete costs of ISD algorithms was performed in [3]. Here the bit complexity estimates for the algorithmic cost of performing the MMT algorithm on nearly all proposed parameter sets are significantly lower than claimed by the submissions. Accordingly, this work raised some discussions (in the NIST PQC forum [31,32]) about whether the currently proposed parameter sets of code based schemes actually match their security levels and if so where to obtain more reliable estimates. We found that the analysis of [3] is flawed for both advanced ISD algorithms that are considered, namely the BJMM and the MMT algorithm. We give a more detailed description of that flaw in Appendix A. Besides that flaw the authors also use techniques that might affect the success probability of the algorithms, but are not mentioned in the latter analysis (as for instance a trimming of lists that exceed certain sizes). However, the analysis of the other ISD variants given in [3] seems to be correct.

In [8] Both and May describe an ISD improvement entirely based on nearest neighbor search. They also consider nearest neighbor algorithms other than May-Ozerov, which was motivated by the non-fulfillment of necessary prerequisites. However, their analysis is purely asymptotical and focuses entirely on the constant error regime.

Our contribution The contribution of this work is twofold. First we uncover relations between major ISD improvements, showing that all of them are build on nearest neighbor search. In the case of the BJMM and MMT algorithms, this view allows us to detect and finally correct deficiencies in the way the nearest neighbor search is performed. Our fix results in two new variants of the BJMM algorithm, which practically (and probably also asymptotically) outperform the original BJMM algorithm. Our work therefore contributes substantially to a better understanding of ISD algorithms.

Moreover, as another contribution, we give a unified framework based on nearest neighbor search, which allows to obtain variants of all major ISD improvements. By an easy exchange of the used nearest neighbor routines we obtain *practical* variants of the improvements by May-Ozerov and Both-May, which were previously disregarded in the context of concrete hardness estimations.

By an analysis of our framework for different instantiations we obtain formulas for the concrete complexity to solve the syndrome decoding problem. We implemented these estimates for all variants considered in this work (and more[1]) and provide the source code in form of an easy to use estimator program (mostly) written in *python*.[2] This allows for an effortless recomputation of our results, estimation of the security levels for newly proposed parameter sets as well as custom modifications if required.

Finally we give the classical estimates for all proposed parameter sets of code-based schemes being part of the third round of the NIST PQC call, namely Classic McEliece [10], BIKE [1] and HQC [23]. Here we consider different memory

[1] The focus of this work lies on advanced algorithms, but our estimator also provides the estimates for asymptotically inferior procedures.

[2] The estimator can be downloaded at `https://github.com/Crypto-TII/syndrome_decoding_estimator`.

access cost models and memory limitations. We find that essentially all parameter sets of the three schemes match their claimed security levels, with a slight outlier in form of the McEliece category three parameter set. Also we provide quantum estimates under the NIST metric of a `maxdepth` constraint, which limits the depth of the quantum circuit. We find that under this constraint even a very optimistic analysis of the quantum version of Prange's ISD algorithm [5] lets all proposed parameter sets match their claimed quantum security level.

The rest of the paper is organized as follows. In Section 2 we give basic definitions and present the practical nearest neighbor algorithms used in our analyses. In Section 3 we line out the nearest neighbor relations between known ISD variants. A reader only interested in concrete hardness estimations may skip this section. Subsequently, in Section 4 we present and analyze the framework and its instantiations to obtain formulas for practical cost estimations. Finally in Section 5 we present our hardness estimates for the classical and quantum security of all proposed parameter sets of McEliece, BIKE and HQC.

2 Preliminaries

We denote vectors as bold lower case letters and matrices with capital letters. We let I_n be the $n \times n$ identity matrix. For an integer $i \in \mathbb{N}$ we define $[i] := \{1, 2, \ldots, i\}$. Let $\mathbf{v} = (v_1, v_2, \ldots v_n)$ be a vector and $S \subseteq [n]$ then we denote by $\mathbf{v}_S$ the projection of $\mathbf{v}$ onto its coordinates indexed by S. For $\mathbf{w} \in \mathbb{F}_2^n$ we define $\mathrm{wt}(\mathbf{w}) := |\{i \in [n] \mid w_i = 1\}|$ to be the Hamming weight of $\mathbf{w}$. Furthermore we let $\mathcal{B}_p^n := \{\mathbf{w} \in \mathbb{F}_2^n \mid \mathrm{wt}(\mathbf{w}) = p\}$ be the set of all binary vectors of length n and Hamming weight p.

Coding Theory A binary linear code $\mathcal{C}$ of length n and dimension k is a k-dimensional subspace of $\mathbb{F}_2^n$. Such a code can be defined as the image of a generator matrix $G \in \mathbb{F}_2^{k \times n}$ or via the kernel of a parity check matrix $H \in \mathbb{F}_2^{(n-k) \times n}$. We use the parity check description of the code throughout this work. Note that since any codeword $\mathbf{c} \in \mathcal{C}$ satisfies $H\mathbf{c} = \mathbf{0}$ the task of decoding a faulty codeword $\mathbf{y} = \mathbf{c} + \mathbf{e}$ for some error $\mathbf{e}$ yields the identity

$$H\mathbf{y} = H(\mathbf{c} + \mathbf{e}) = H\mathbf{e} =: \mathbf{s} \ .$$

The vector $\mathbf{s}$ is usually called the *syndrome* of $\mathbf{y}$, while obtaining $\mathbf{e}$ from given H and $\mathbf{s}$ is called the *syndrome decoding problem*.

Definition 2.1 (Syndrome Decoding Problem). *Let $H \in \mathbb{F}_2^{(n-k) \times n}$ be the parity check matrix of a random linear code, $\mathbf{s} \in \mathbb{F}_2^{n-k}$ and $\omega \in [n]$. The* syndrome decoding problem *asks to find a vector $\mathbf{e} \in \mathbb{F}_2^n$ with $\mathrm{wt}(\mathbf{e}) = \omega$ satisfying $H\mathbf{e} = \mathbf{s}$.*

Nearest Neighbor At the heart of all algorithms presented in this work lies a specific kind of nearest neighbor or approximate matching problem, which we define in the following.

Definition 2.2 (Approximate Matching Problem). *Let $M \in \mathbb{F}_2^{r \times m}$ be a random matrix, $\mathbf{s} \in \mathbb{F}_2^r$ and $\delta, p \in \mathbb{N}$. The* approximate matching problem *asks to find all solutions $\mathbf{e} \in \mathbb{F}_2^m$ satisfying*

$$\mathrm{wt}(\mathbf{e}) = p \text{ and } \mathrm{wt}(M\mathbf{e} + \mathbf{s}) = \delta.$$

We write

$$M\mathbf{e} \approx_\delta \mathbf{s} \ ,$$

to denote that $M\mathbf{e}$ matches $\mathbf{s}$ on all but δ coordinates and call this equation an approximate matching identity.

Usually, routines to solve the approximate matching problem exploit a direct reduction to the bichromatic nearest neighbor problem. In this problem we are given two lists of binary vectors and are asked to find all pairs between those lists with distance δ. Therefore split $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ in the sum of two vectors. For now let us consider a meet-in-the-middle split and even m (without loss of generality), where $\mathbf{e}_1 = (\mathbf{d}_1, 0^{\frac{m}{2}})$ and $\mathbf{e}_2 = (0^{\frac{m}{2}}, \mathbf{d}_2)$ with $\mathbf{d}_1, \mathbf{d}_2 \in \mathcal{B}_{p/2}^{m/2}$, but also other splittings are possible.[3] Then all $\mathbf{e}_1$, respectively $\mathbf{e}_2$, are enumerated and $M\mathbf{e}_1$ is stored in list L_1, respectively $M\mathbf{e}_2 + \mathbf{s}$ is stored in list L_2. Now, a pair with distance δ between those lists fulfills by definition the approximate matching identity

$$M(\mathbf{e}_1 + \mathbf{e}_2) \approx_\delta \mathbf{s} \ .$$

Also note that due to the chosen splitting $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ has weight p by construction, as desired.

The asymptotically fastest known algorithm for solving the bichromatic nearest neighbor problem where the lists are of size $\tilde{\mathcal{O}}\left(2^{c \cdot m}\right)$, for constant c, is by May-Ozerov [22]. While the algorithm achieves theoretically close to optimal complexities [19], it inherits a huge polynomial overhead limiting its practicality, despite recent efforts to reduce that overhead [13]. As one of the major goals of this work is to provide precise *practical* estimates we do not consider the algorithm by May-Ozerov. However, we use different, simpler but more practical approaches.

Let us briefly outline three techniques to solve the bichromatic nearest neighbor problem, where we measure all running times in vector operations in $\mathbb{F}_2^m$. The most basic variant is a naive enumeration, which we call BRUTEFORCE in the following. The algorithm simply enumerates all pairs of $L_1 \times L_2$ and checks if their distance is δ. Clearly this algorithm has running time

$$T_{\mathrm{B}} = |L_1 \times L_2|.$$

[3] Note that this splitting only allows to construct balanced solutions $\mathbf{e}$, which have exactly weight $\frac{p}{2}$ on the upper and lower half. While we take this into account when deriving our concrete estimates let us neglect this fact for now.

Meet-in-the-Middle. A slightly more sophisticated algorithm uses a meet-in-the-middle approach. First, the lists are enlarged by factor of the number of possible vectors of Hamming weight equal to δ. Thus, for every possible such vector, its sum with the original element is present in the enlarged list. To balance the complexities the δ-Hamming weight vector is again split in a meet-in-the-middle fashion among both lists. This implies that the algorithm can only find pairs of elements whose distance is balanced, i.e. splits equally on both sides of their sum. We will take this into account in our precise analysis later, but for know let us ignore this issue. The pseudocode of the algorithm is given in Algorithm 1. Note that after the addition of the $\delta/2$-weighted vectors the nearest neighbor problem degenerates to a search for equality.

Algorithm 1 MEET-IN-THE-MIDDLE

Input: $L_1, L_2 \in (\mathbb{F}_2^m)^*$, $\delta \in [m]$
Output: all $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ with $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \delta$

1: $L_1' = \{\mathbf{x} + (\mathbf{d}, \mathbf{0}) \mid \mathbf{x} \in L_1,\ \mathbf{d} \in \mathcal{B}_{\delta/2}^{m/2}\}$
2: $L_2' = \{\mathbf{y} + (\mathbf{0}, \mathbf{d}) \mid \mathbf{y} \in L_2,\ \mathbf{d} \in \mathcal{B}_{\delta/2}^{m/2}\}$
3: **for** $\mathbf{y}' \in L_2'$ **do**
4: $\quad L \leftarrow L \cup \{(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}' = \mathbf{y}', \mathbf{x}' \in L_1'\}$
5: **return** L

Therefore note that for every pair $(\mathbf{x}, \mathbf{y}) \in L$ it holds that

$$\mathbf{x}' = \mathbf{y}' \Leftrightarrow \mathbf{x} + (\mathbf{d}_1, \mathbf{0}) = \mathbf{y} + (\mathbf{0}, \mathbf{d}_2) \Leftrightarrow \mathbf{x} + \mathbf{y} = (\mathbf{d}_1, \mathbf{d}_2) \Rightarrow \mathrm{wt}(\mathbf{x}+\mathbf{y}) = \delta \ .$$

The lists L_1' and L_2' are of size $\binom{m/2}{w/2} \cdot |L_1|$, while the output list L is of expected size $|L_1' \times L_2'|/2^m$. As we only need to search for equality the time complexity to construct L is linear in these lists sizes,[4] which gives a time complexity of

$$T_{\text{MITM}} = 2 \cdot \binom{m/2}{w/2} \cdot |L_1| + \frac{|L_1|^2 \binom{m/2}{w/2}^2}{2^m} \ . \tag{1}$$

Indyk-Motwani. The third routine we consider for solving the nearest neighbor problem is based on locality sensitive hashing introduced by Indyk and Motwani [16]. Let $\mathbf{z} = \mathbf{x} + \mathbf{y}$ for $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ be an element with weight δ. Then the algorithm guesses $\lambda \in \mathbb{N}$ coordinates $I \subset [m]$ of $\mathbf{z}$ for which it assumes that $\mathbf{z}_I = \mathbf{0}$ holds. Now, an exact matching on these λ coordinates is performed between L_1 and L_2. For each match $(\mathbf{x}', \mathbf{y}')$, the algorithm then checks if $\mathrm{wt}(\mathbf{z}') = \delta$ for $\mathbf{z}' = \mathbf{x}' + \mathbf{y}'$ holds.

The algorithm relies on the fact that for elements whose sum has small weight, the projection to one of the coordinates of those elements is more likely to be equal then for elements whose sum has larger weight. Algorithm 2 gives a pseudocode description of the procedure.

[4] By using a hashing strategy.

Algorithm 2 INDYK-MOTWANI

Input: $L_1, L_2 \in (\mathbb{F}_2^m)^*$, $\delta \in [m]$
Output: all $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ with $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \delta$

1: $\lambda := \min(\log L_1, m - 2\delta)$, $N := \frac{\binom{m-\lambda}{\delta}}{\binom{m}{\delta}}$
2: **for** $i = 1$ **to** N **do**
3: choose random $I \subseteq [m]$ with $|I| = \lambda$
4: **for** $(\mathbf{x}, \mathbf{y}) \in \{(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2 \mid \mathbf{x}_I = \mathbf{y}_I\}$ **do**
5: **if** $\mathrm{wt}(\mathbf{x}+\mathbf{y}) = \delta$ **then**
6: $L \leftarrow L \cup (\mathbf{x}, \mathbf{y})$
7: **return** L

Note that the probability that for a $\mathbf{z} \in \mathbb{F}_2^m$ with $\mathrm{wt}(\mathbf{z}) = \delta$ the projection to a random choice of λ distinct coordinates is the zero vector is

$$p := \Pr\left[\mathbf{z}_I = 0^\lambda \mid \mathbf{z} \in \mathbb{F}_2^m \wedge \mathrm{wt}(\mathbf{z}) = \delta\right] = \frac{\binom{m-\lambda}{\delta}}{\binom{m}{\delta}} \ .$$

Similar to the meet-in-the-middle approach the time for the construction of L is linear in the involved lists sizes, which results in an expected time complexity of

$$T_{\mathrm{IM}} = p^{-1} \cdot (2|L_1| + |L_1|^2/2^\lambda) = \frac{\binom{m}{\delta} \cdot (2|L_1| + |L_1|^2/2^\lambda)}{\binom{m-\lambda}{\delta}} \tag{2}$$

An approximation of the binomial coefficients via Stirling's formula and analytical analysis yields a global minimum at $\lambda = \min(\log |L_1|, m - 2\delta)$. Numerical computations show, that this value is also very close to the optimum when instead considering the more precise form of the runtime formula given in in Equation (2).

3 ISD algorithms from a nearest neighbor perspective

Let us start with the ISD algorithm by Prange, which forms the foundation for all advanced techniques. The algorithm first applies a random permutation to the columns of the parity check matrix H. Note that for any permutation matrix $P \in \mathbb{F}_2^{n \times n}$ we have $(HP)(P^{-1}\mathbf{e}) = \mathbf{s}$. Now, by applying Gaussian elimination to its rows we transform HP into *systematic form*, modelled by the multiplication with an invertible matrix $Q \in \mathbb{F}_2^{(n-k)\times(n-k)}$

$$QHP = \left(\tilde{H}\ I_{n-k}\right), \text{ where } \tilde{H} \in \mathbb{F}_2^{(n-k)\times(k)}. \tag{3}$$

Note that for a random permutation matrix there exists such an invertible Q with constant probability. Further let $(P^{-1}\mathbf{e}) = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$ and $Q\mathbf{s} = \tilde{\mathbf{s}} \in \mathbb{F}_2^{n-k}$, then the following identity holds

$$Q(HP)(P^{-1}\mathbf{e}) = (\tilde{H}\mathbf{e}' + \mathbf{e}'') = \tilde{\mathbf{s}}.$$

Assume that the permutation P induces a weight distribution of

$$\mathrm{wt}(\mathbf{e}') = p \text{ and hence } \mathrm{wt}(\mathbf{e}'') = \omega - p \ , \tag{4}$$

then it suffices to find an $\mathbf{e}' \in \mathbb{F}_2^k$ of weight p satisfying

$$\mathrm{wt}(\tilde{H}\mathbf{e}' + \tilde{\mathbf{s}}) = \mathrm{wt}(\mathbf{e}'') = \omega - p \ . \tag{5}$$

Once a suitable permutation P and a vector $\mathbf{e}'$ are found $\mathbf{e} = P(\mathbf{e}', \tilde{H}\mathbf{e}' + \tilde{\mathbf{s}})$ forms a solution to the syndrome decoding problem. Note that Equations (4) and (5) yield a approximate matching identity according to Definition 2.2

$$\tilde{H}\mathbf{e}' \approx_{\omega-p} \tilde{\mathbf{s}} \ . \tag{6}$$

While Prange's algorithm chooses the weight p of $\mathbf{e}'$ equal to zero and thus does not have to put any effort into solving the approximate matching problem,[5] all further improvements choose $p > 0$.

Choosing $p > 0$ and applying the bruteforce approach that simply enumerates all possible $\mathbf{e}'$ of weight p yields a polynomial improvement due to Lee and Brickell [20].

Applying instead the Indyk-Motwani or the meet-in-the-middle approaches results in algorithmic analogs of the well-known ISD improvements by Stern [29] and Dumer [11] respectively.

Stern's original algorithm. Stern [29] improved on the approach of Prange by introducing an ℓ-window of zero entries in the error $\mathbf{e}''$. Thus one considers $(P^{-1}\mathbf{e}) = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^k \times \mathbb{F}_2^{n-k}$, where $\mathbf{e}'' = (0^\ell, \tilde{\mathbf{e}})$. Note that due to this weight distribution $\tilde{H}\mathbf{e}'$ matches the syndrome on the first ℓ coordinates, as we have

$$\tilde{H}\mathbf{e}' = \tilde{\mathbf{s}} + \mathbf{e}'' = \tilde{\mathbf{s}} + (0^\ell, \tilde{\mathbf{e}}) = (\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_2 + \tilde{\mathbf{e}}) \ ,$$

where $\tilde{\mathbf{s}} = (\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_2) \in \mathbb{F}_2^\ell \times \mathbb{F}_2^{n-k-\ell}$. Now Stern's algorithm uses the identity $(\tilde{H}\mathbf{e}')_{[\ell]} = \tilde{\mathbf{s}}_1$ to perform a meet-in-the-middle search on $\mathbf{e}'$. Thus, it splits $\mathbf{e}' = (\mathbf{e}_1, 0^{\frac{k}{2}}) + (0^{\frac{k}{2}}, \mathbf{e}_2)$, constructs two lists containing $\tilde{H}(\mathbf{x}, 0^{\frac{k}{2}})$ and $\tilde{H}(0^{\frac{k}{2}}, \mathbf{y})$ respectively for all $\mathbf{x}, \mathbf{y} \in \mathcal{B}_{p/2}^{k/2}$ and searches between those lists for pairs that sum to $\tilde{\mathbf{s}}_1$ on their first ℓ coordinates. For every matching pair $((\mathbf{x}', \mathbf{0}), (\mathbf{0}, \mathbf{y}'))$ it is checked if $\tilde{H}(\mathbf{x}', \mathbf{y}') + \tilde{\mathbf{s}}$ has weight $\omega - p$, which is particularly satisfied when $(\mathbf{x}', \mathbf{y}') = (\mathbf{e}_1, \mathbf{e}_2)$.

NN-perspective of Stern's Algorithm. Let us modify the permutation step by first aiming for the weight distribution of Prange (Equation (4)) and then permuting the error $\mathbf{e}''$ separately several times until we may expect the desired ℓ-window. For every such permutation we continue with the meet-in-the-middle step of Stern's algorithm. First note that this does not change the expected amount of necessary permutations until the weight is distributed as in Stern's original algorithm. However, it shows that the algorithm by Stern is actually solving the approximate matching identity from Equation (6) via Indyk-Motwani. Here the matching on the first ℓ coordinates after the permutation corresponds to the matching on a random projection of ℓ coordinates used by Indyk-Motwani.

[5] For $p = 0$ a simple check if $\mathrm{wt}(\tilde{\mathbf{s}}) = \omega$ suffices to determine if the permutation distributes the weight correctly.

Dumer's original algorithm Dumer [11] changed the procedure by increasing the dimension of $\mathbf{e}'$ to $k+\ell$, which allowed him to get rid of the ℓ-window of zeros in the permutation. Therefore he defines the systematic form as

$$QHP = \begin{pmatrix} H_1 & \mathbf{0} \\ H_2 & I_{n-k-\ell} \end{pmatrix}, \tag{7}$$

where $H_1 \in \mathbb{F}_2^{\ell \times (k+\ell)}$ and $H_2 \in \mathbb{F}_2^{(n-k-\ell)\times(k+\ell)}$. Again it is aimed for a weight distribution where for $P^{-1}\mathbf{e} = (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{n-k-\ell}$ it holds $\mathrm{wt}(\mathbf{e}') = p$, and $\mathrm{wt}(\mathbf{e}'') = \omega - p$. Due to the increased dimension of $\mathbf{e}'$ we get

$$QHP(\mathbf{e}', \mathbf{e}'') = (H_1\mathbf{e}', H_2\mathbf{e}' + \mathbf{e}'') = (\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_2) = \tilde{\mathbf{s}} \ ,$$

where $(\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_2) \in \mathbb{F}_2^{\ell} \times \mathbb{F}_2^{n-k-\ell}$. The algorithm then uses the identity on the first ℓ bits again to search for $\mathbf{e}'$ using a meet-in-the-middle strategy. Again for each computed candidate $\mathbf{x}$ for $\mathbf{e}'$ satisfying the identity it checks if $\mathrm{wt}(H_2\mathbf{x} + \tilde{\mathbf{s}}_2) = \mathrm{wt}(\mathbf{e}'') = \omega - p$ and if so outputs the solution $P(\mathbf{x}, H_2\mathbf{x} + \tilde{\mathbf{s}}_2)$.

NN-perspective of Dumer's algorithm Note that the original matrix used by Prange's algorithm (see Equation (3)) is already in systematic form according to the definition of Dumer, since

$$\begin{pmatrix} \tilde{H} & I_{n-k} \end{pmatrix} = \begin{pmatrix} \tilde{H}_1 & I_\ell & \mathbf{0} \\ \tilde{H}_2 & \mathbf{0} & I_{n-k-\ell} \end{pmatrix}, \text{ where } \tilde{H} \in \mathbb{F}_2^{(n-k)\times(k)}.$$

Thus we obtain Equation (7) by setting $H_1 = \begin{pmatrix} \tilde{H}_1 & I_\ell \end{pmatrix}$ and $H_2 = \begin{pmatrix} \tilde{H}_2 & \mathbf{0} \end{pmatrix}$. Now let us further split $\mathbf{e}' \in \mathbb{F}_2^{k+\ell}$ in two parts $\mathbf{e}' = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_2^k \times \mathbb{F}_2^\ell$ and reconsider the identity

$$H_1\mathbf{e}' = \tilde{\mathbf{s}}_1 \ ,$$

which now becomes

$$H_1\mathbf{e}' = \begin{pmatrix} \tilde{H}_1 & I_\ell \end{pmatrix} (\mathbf{e}_1, \mathbf{e}_2) = \tilde{H}_1\mathbf{e}_1 + \mathbf{e}_2 = \tilde{\mathbf{s}}_1 \ .$$

Thus we are facing again a nearest neighbor identity $\tilde{H}_1\mathbf{e}_1 \approx_{|\mathbf{e}_2|} \tilde{\mathbf{s}}_1$. Dumer's algorithm enforces only a joined weight distribution of $\mathrm{wt}((\mathbf{e}_1, \mathbf{e}_2)) = p$ and, hence, we do not exactly know the weight of the approximate matching problem we are facing. However, with inverse polynomial probability the weight distributes proportionally on both sides, giving $\mathrm{wt}(\mathbf{e}_1) = \frac{k \cdot p}{k+\ell}$ and $\mathrm{wt}(\mathbf{e}_2) = \frac{\ell \cdot p}{k+\ell}$. Now using the Meet-in-the-Middle algorithm to solve this instance we obtain a version of Dumer's algorithm from a nearest neighbor perspective achieving the same asymptotic complexity.

A natural question which comes to mind when considering the nearest neighbor version of Dumer is: why should it be optimal to choose a joined weight distribution for $(\mathbf{e}_1, \mathbf{e}_2)$? By introducing two different weight parameters p_1 and p_2 for both sides of $\mathbf{e}'$ we obtain an algorithmic analogue of the improvement of Bernstein et al. [7] known as Ball Collision Decoding (BCD), which slightly improves on Dumer's algorithm.

Also one might question the optimality of Stern's procedure, which performs the nearest neighbor search on the whole $n-k$ coordinates of $\mathbf{e}''$ and weight $\omega - p$ instead of using a reduced instance of length ℓ_2 and weight p_2, like the BCD variant. We found that refining Stern's algorithm using the additional parameters ℓ_2 and p_2 yields a slight improvement similar to the BCD one.

The MMT algorithm Let us now turn our focus to the ISD improvements by May, Meurer and Thomae (MMT) [21] as well as by Becker, Joux, May and Meurer (BJMM) [4]. In a nutshell these algorithms first apply a permutation, similar to the previous algorithms. However, instead of splitting the vector $\mathbf{e}'$ in two addends, as done by Dumer, Stern or BCD, they split it in four (MMT) or eight (BJMM). Then all candidates for the addends are enumerated in a meet-in-the-middle fashion. A binary search tree (similar to Wagner's k-tree algorithm [33]) is subsequently used to construct the solution as sum of base list elements. Additionally, they do not use a disjoint splitting of the addends throughout the tree, which gives several different *representations* of the solution. For example the MMT algorithm represents $\mathbf{e}' = \mathbf{e}_1 + \mathbf{e}_2$ with $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^n$ and then splits $\mathbf{e}_1$ and $\mathbf{e}_2$ in a meet-in-the-middle fashion (as before). This gives several different combinations of $(\mathbf{e}_1, \mathbf{e}_2)$ that sum up to $\mathbf{e}'$. As the binary search tree imposes restrictions on the exact form of the solution, a careful choice of parameters lets a single of these representations fulfill the constraints. Note that the knowledge of a single representation of $\mathbf{e}'$ suffices to solve the syndrome decoding problem.

As the structure of the BJMM and MMT algorithm is quite similar we stick with a description of the MMT algorithm for now, highlighting their differences later.

Let us explain the algorithm in a bit more detail. The MMT (as well as the BJMM) algorithm uses the same preprocessing step as Dumer, hence H is in systematic form according to Equation (7) and the weight similarly distributes on $P^{-1}\mathbf{e} := (\mathbf{e}', \mathbf{e}'') \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{n-k-\ell}$ as $\mathrm{wt}(\mathbf{e}') = p$ and $\mathrm{wt}(\mathbf{e}'') = \omega - p$. Now, as mentioned before, the algorithm splits

$$\mathbf{e}' = \underbrace{(\mathbf{a}_1, 0^{\frac{k+\ell}{2}}) + (0^{\frac{k+\ell}{2}}, \mathbf{a}_2)}_{\mathbf{e}_1} + \underbrace{(\mathbf{a}_3, 0^{\frac{k+\ell}{2}}) + (0^{\frac{k+\ell}{2}}, \mathbf{a}_4)}_{\mathbf{e}_2} \ ,$$

with $\mathbf{a}_i \in \mathcal{B}_{p/4}^{(k+\ell)/2}, i = 1, 2, 3, 4$, hence, by construction we have $\mathbf{e}_1, \mathbf{e}_2 \in \mathcal{B}_{p/2}^{k+\ell}$.

Next candidates for $\mathbf{a}_i$ are constructed by enumerating all possible values from $\mathcal{B}_{p/4}^{(k+\ell)/2}$ in the base lists L_i. Now, one chooses some random $\mathbf{t} \in \mathbb{F}_2^{\ell_1}$, for some optimized $\ell_1 \leq \ell$, and constructs a new list L_{12} by just considering those elements $(\mathbf{x}, \mathbf{y}) \in L_1 \times L_2$ for which it holds that

$$(H_1(\mathbf{x} + \mathbf{y}))_{[\ell_1]} = \mathbf{t} \ .$$

Now the same is done for the lists L_3 and L_4, thus they are merged in a new list L_{34} but using a modified target $\mathbf{t}' = \mathbf{t} + (\tilde{\mathbf{s}}_1)_{[\ell_1]}$. This choice of $\mathbf{t}'$ ensures that $(\mathbf{v}, \mathbf{w}) \in L_{12} \times L_{34}$ satisfy:

$$H_1(\mathbf{v} + \mathbf{w}) = (\tilde{\mathbf{s}}_1)_{[\ell_1]} \ ,$$

hence the desired identity is already matched on the lower ℓ_1 coordinates. Finally the algorithm merges lists L_{12} and L_{34} in a list L_{1234} by enforcing the identity on all ℓ bits and then checks if for any $\mathbf{z} \in L_{1234}$ it holds that $\mathrm{wt}(\tilde{H}_2\mathbf{z} + \tilde{\mathbf{s}}_2) = \mathrm{wt}(\mathbf{e}'') = \omega - p$ and, if so, outputs the solution.

NN-perspective and an algorithmic shortcoming of the MMT algorithm. We show in the following that the MMT algorithm also uses a meet-in-the-middle strategy for solving nearest-neighbor equations. But contrary to the procedure given in Algorithm 1, too many vectors are enumerated in the base lists, which unnecessarily increases the list sizes and results in undesired list distributions for special inputs.

Similar to the NN-perspective of Dumer's algorithm, let H be in systematic form as given by Equation (3), which is

$$H = \left(\tilde{H}\ I_{n-k}\right) = \begin{pmatrix} \tilde{H}_1 & I_\ell & \mathbf{0} \\ \tilde{H}_2 & \mathbf{0} & I_{n-k-\ell} \end{pmatrix}, \text{ where } \tilde{H} \in \mathbb{F}_2^{(n-k)\times(k)}.$$

Additionally, let $\mathbf{e}' = (\mathbf{e}'_1, \mathbf{e}'_2, \mathbf{e}'_3) \in \mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell-\ell_1}$ and let the weight on each of the $\mathbf{e}'_i$ be $p_i := \frac{|\mathbf{e}'_i|\cdot p}{k+l}$. Also, for now consider the base list elements of the MMT algorithm to be formed as

$$\left(\mathcal{B}_{p_1/4}^{k/2} \times 0^{\frac{k}{2}} \times \mathcal{B}_{p_2/4}^{\ell_1/2} \times 0^{\frac{\ell_1}{2}} \times \mathcal{B}_{p_3/4}^{(\ell-\ell_1)/2} \times 0^{\frac{\ell-\ell_1}{2}}\right) \text{ and}$$
$$\left(0^{\frac{k}{2}} \times \mathcal{B}_{p_1/4}^{k/2} \times 0^{\frac{\ell_1}{2}} \times \mathcal{B}_{p_2/4}^{\ell_1/2} \times 0^{\frac{\ell-\ell_1}{2}} \times \mathcal{B}_{p_3/4}^{(\ell-\ell_1)/2}\right) ,$$

rather than

$$\left(\mathcal{B}_{p/4}^{(k+l)/2} \times 0^{\frac{k+\ell}{2}}\right) \text{ and } \left(0^{\frac{k+\ell}{2}} \times \mathcal{B}_{p/4}^{(k+l)/2}\right) .$$

Thus, each of the $\mathbf{e}'_i$ is getting enumerated in a meet-in-the-middle fashion in the base lists.[6] Additionally let us write H_1 as

$$H_1 := \left(\tilde{H}_1\ I_\ell\right) = \begin{pmatrix} \tilde{H}_{11} & I_{\ell_1} & \mathbf{0} \\ \tilde{H}_{12} & \mathbf{0} & I_{\ell-\ell_1} \end{pmatrix}$$

Now let us consider the first join of base lists. For this join only elements $\mathbf{x} \in L_1$ and $\mathbf{y} \in L_2$ are considered for which

$$\left(H_1(\mathbf{x}+\mathbf{y})\right)_{[\ell_1]} = \mathbf{t} \ . \tag{8}$$

By letting $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \in \mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell-\ell_1}$ and $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \mathbf{y}_3)$, analogously, Equation (8) becomes

$$\begin{aligned} & \tilde{H}_{11}(\mathbf{x}_1 + \mathbf{y}_1) + \mathbf{x}_2 + \mathbf{y}_2 = \mathbf{t} \\ \Leftrightarrow \quad & \tilde{H}_{11}(\mathbf{x}_1 + \mathbf{y}_1) = \mathbf{t} + \mathbf{x}_2 + \mathbf{y}_2 \ . \end{aligned} \tag{9}$$

[6] Note that both sets differ only in a polynomial fraction of their elements. Furthermore our argumentation also holds when using the original base lists, but the refinement allows for easier illustration by yielding approximate matching identities with fixed distances.

Note that by construction $\mathrm{wt}(\mathbf{x}_1 + \mathbf{y}_1) = p_1/2$ and $\mathrm{wt}(\mathbf{x}_2 + \mathbf{y}_2) = p_2/2$, hence the newly constructed list L_{12} consists only of vectors having weight $p_1/2$, which are $p_2/2$ close to $\mathbf{t}$ when multiplied by $\tilde{H}_{11}$. Since all possible combinations of $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{y}_1, \mathbf{y}_2)$ are considered in a meet-in-the-middle fashion, this join solves the approximate matching identity

$$H\mathbf{x} \approx_{p_2/2} \mathbf{t} \ ,$$

for $\mathbf{x}$ with $\mathrm{wt}(\mathbf{x}) = p_1/2$.

Contrary to the Meet-in-the-Middle algorithm from Section 2 the MMT algorithm additionally enumerates all values for $\mathbf{x}_3$ (resp. $\mathbf{y}_3$) in the base lists even though they are not taken into account by the matching routine. Thus, whenever any element satisfies Equation (9), it is added to L_{12} for any combination of $(\mathbf{x}_3, \mathbf{y}_3) \in \mathcal{B}_{p_3/4}^{(\ell-\ell_1)/2} \times \mathcal{B}_{p_3/4}^{(\ell-\ell_1)/2}$.

Thus, if $(\mathbf{z}_1, \mathbf{z}_2) = (\mathbf{x}_1 + \mathbf{y}_1, \mathbf{x}_2 + \mathbf{y}_2)$ describes an element satisfying Equation (9), all elements of

$$\left\{ (\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3) \mid \mathbf{z}_3 \in \left(\mathcal{B}_{p_3/4}^{(\ell-\ell_1)/2} \times \mathcal{B}_{p_3/4}^{(\ell-\ell_1)/2} \right) \right\} \tag{10}$$

are added to L_{12} (analogously the same holds for L_{34}).

The final join then solves the nearest neighbor identity on the upper $\ell - \ell_1$ bits for target $(\tilde{\mathbf{s}}_1)_{[\ell_1+1,\ell]}$ and distance p_3. But instead of using a disjoint split of the vectors with weight p_3, as done by Algorithm 1, the weight is distributed over the full $\ell - \ell_1$ coordinates (see $\mathbf{z}_3$ of Equation (10)). Thus, there exist multiple different representations for every possible difference, resulting in as many duplicates being added to L_{1234} for every element fulfilling the approximate matching identity on the upper bits.

Not only would a single representation of the solution in L_{1234} suffice to solve the problem, but imagine there would be a subsequent level in the tree, which is e.g. the case for the BJMM algorithm. Then the degenerated list distribution would significantly affect the list sizes of following levels and, implicitly, the time complexity and correctness of the algorithm. We want to stress that this problem only occurs if the algorithm is provided with a parity-check matrix H as defined in Equation (3). If the input matrix has the shape given in Equation (7) with random H_1 this seems to re-randomize the duplicates such that the list sizes match their expectations, as experiments have shown [2, 12]. Nevertheless, it enables us in the next section to improve on the standard MMT (respectively BJMM) algorithm by changing the way the base lists are constructed.

Other advanced ISD variants. Let us briefly outline the differences between the MMT algorithm and the improvements made by Becker-Joux-May-Meurer [4], May-Ozerov [22] and Both-May [8]. The BJMM algorithm works similar to the MMT algorithm but increases the weight of the candidates for the $\mathbf{a}_i$ to $p/4 + \varepsilon$. The parameter ε then accounts for ones that cancel during addition. While increasing list sizes, this also increases the amount of representations allowing for larger constraint choices (the length of ℓ_1) when constructing lists of

subsequent levels. Additionally, the increased amount of representations yields a theoretically optimal search tree depth of three (instead of two), to cancel out the representations and balance the tree most effectively. The ideas introduced with the BJMM algorithm were adopted by both May-Ozerov and Both-May variants. May and Ozerov then exchanged the meet-in-the-middle strategy to solve the nearest neighbor problem on the last level by their own more efficient algorithm for nearest neighbor search. Both and May finally exploited the nearest neighbor search technique from May-Ozerov for the construction of all lists of the tree.

4 An ISD framework based on nearest neighbor search

In this section we describe an algorithmic framework for ISD algorithms based explicitly on nearest neighbor search that resolves the shortcomings mentioned in the previous section. We are able to obtain variants of all major ISD improvements by choosing specific configurations of our framework. Additionally, we can easily exchange costly routines, such as May-Ozerov nearest neighbor search by more practical algorithms. Similar to the MMT algorithm, our framework uses a search tree to construct the solution. To obtain the lists of each level, nearest neighbor search is exploited. The framework then yields the basis for obtaining our practical security estimates.

Remark 4.1. Our complexity estimates show that, for the cryptographically interesting error regimes, a search tree depth of two is (almost) optimal, regardless of the chosen instantiation of the framework. We find that this is the case in memory constrained and unconstrained settings, as well as under consideration of different memory access costs. Only in some rare cases, an increase to depth three gives minor improvements of a factor strictly less than two (for the proposed parameter sets of McEliece, BIKE and HQC). Hence, for didactic reasons we describe our framework only in depth two.

Let us assume the parity check matrix is in systematic form according to Equation (3) and let us write the matrix as

$$H = \left(\tilde{H}\ I_{n-k}\right) = \begin{pmatrix} \tilde{H}_1 & I_{\ell_1} & \mathbf{0} & \mathbf{0} \\ \tilde{H}_2 & \mathbf{0} & I_{\ell_2} & \mathbf{0} \\ \tilde{H}_3 & \mathbf{0} & \mathbf{0} & I_{\ell_3} \end{pmatrix}, \text{ where } \tilde{H} \in \mathbb{F}_2^{(n-k)\times k}\ , \tag{11}$$

and let

$$\tilde{\mathbf{s}} := (\tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_2, \tilde{\mathbf{s}}_3) \in \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell_2} \times \mathbb{F}_2^{\ell_3} \tag{12}$$

be the corresponding syndrome (after the Gaussian elimination). The permutation is assumed to distribute the weight on $\mathbf{e} = (\mathbf{e}', \mathbf{e}_1'', \mathbf{e}_2'', \mathbf{e}_3'') \in (\mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell_2} \times \mathbb{F}_2^{\ell_3})$ as

$$\mathrm{wt}(\mathbf{e}') = p \text{ and } \mathrm{wt}(\mathbf{e}_i'') = \omega_i \text{ for } i = 1, 2, 3, \tag{13}$$

where $\ell_1, \ell_2, \omega_1, \omega_2$ and p are optimized numerically and $\ell_3 := n - k - \ell_1 - \ell_2$, as well as $\omega_3 := \omega - p - \omega_1 - \omega_2$. Note that, by our formulation, the following three

approximate matching identities hold[7]

$$\tilde{H}_i \mathbf{e}' \approx_{\omega_i} \tilde{\mathbf{s}}_i \text{ for } i = 1, 2, 3. \tag{14}$$

Again, we split $\mathbf{e}' = \mathbf{e}_1 + \mathbf{e}_2$ in two addends $\mathbf{e}_i \in \mathbb{F}_2^k$ with $\mathrm{wt}(\mathbf{e}_i) = p_1$ for some numerically optimized p_1.

In the base lists of the search tree (compare also to Figure 1), we enumerate all candidates for $\mathbf{e}_1$, respectively $\mathbf{e}_2$, in a meet-in-the-middle fashion. To obtain the lists of the middle level, we combine two base lists by searching for pairs $(\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2$, respectively $(\mathbf{x}_3, \mathbf{x}_4) \in L_3 \times L_4$, fulfilling the identities

$$\begin{aligned} &\tilde{H}_1(\mathbf{x}_1 + \mathbf{x}_2) \approx_{\omega_{11}} 0^{\ell_1} \text{ and respectively} \\ &\tilde{H}_1(\mathbf{x}_3 + \mathbf{x}_4) \approx_{\omega_{11}} \tilde{\mathbf{s}}_1 \ , \end{aligned}$$

where ω_{11} is another parameter that has to be optimized numerically.

All resulting candidates for $\mathbf{e}_1$ and $\mathbf{e}_2$, namely $\mathbf{x}_{12} = \mathbf{x}_1 + \mathbf{x}_2$ and $\mathbf{x}_{34} = \mathbf{x}_3 + \mathbf{x}_4$, satisfying the above identities are stored in the lists L_{12} and L_{34} respectively. Finally, those two lists are merged in the list L_{1234} by finding all solutions to the identity

$$\tilde{H}_2(\mathbf{x}_{12} + \mathbf{x}_{34}) \approx_{\omega_2} \tilde{\mathbf{s}}_2 \ .$$

Eventually every element of L_{1234} is checked for yielding a solution.

We measure the time complexity of the proposed framework in vector additions in $\mathbb{F}_2^n$. Even though some of the used labels and vectors could be implemented using less than n coordinates, each addition contributes as one. On the one hand this simplifies the analysis and on the other hand it is highly implementation dependent if a vector is indeed implemented using less coordinates

Analysis of the Framework. Let us first analyze our framework. Later, we then compute the concrete complexity for different configurations. Let us start with the correctness. Assume the permutation distributes the error weight as desired (compare to Equation (11)). Now consider the possible decomposition of $\mathbf{e}' = \mathbf{e}_1 + \mathbf{e}_2$ with $\mathrm{wt}(\mathbf{e}_i) = p_1$ and denote the amount of different such representations as R. Furthermore let the probability that any such representation fulfills the restrictions imposed by L_{12} and L_{34} be

$$q := \Pr\left[\mathrm{wt}(\tilde{H}_1 \mathbf{e}_1) = \mathrm{wt}(\tilde{H}_1 \mathbf{e}_2 + \tilde{\mathbf{s}}_1) = \omega_{11} \mid \mathbf{e}' = \mathbf{e}_1 + \mathbf{e}_2\right] \ .$$

Note that the computation of L_{1234} does not impose further constraints on the representations since, by Equation (14), we already conditioned on

$$\tilde{H}_2(\mathbf{e}_1 + \mathbf{e}_2) = \tilde{H}_2 \mathbf{e}' \approx_{\omega_2} \tilde{\mathbf{s}}_2 \ .$$

[7] Note that the equation for $i = 3$ will not be used by the algorithm. It just enables us to perform the nearest neighbor search for $i = 2$ on a reduced sub-instance with flexible ℓ_2, ω_2; instead of being forced to operate always on the full $n - k - \ell_1$ coordinates with weight $\omega - p - \omega_1$.

Algorithm 3 ISD-NN-Framework

Input: parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, syndrome $\mathbf{s}$, error weight $\omega \in [n]$
Output: $\mathbf{e} \in \mathbb{F}_2^n \colon H\mathbf{e} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) = \omega$
Optimize: $\ell_1, \ell_2, \omega_1, \omega_2, \omega_{11}, p_1, p$

1: Let $\tilde{H}_1, \tilde{H}_2, \tilde{\mathbf{s}}_1, \tilde{\mathbf{s}}_2$ and all parameters be as defined in Equations (11) to (13)
2: **repeat**
3: choose random permutation matrix $P \in \mathbb{F}_2^{n\times n}$
4: Transform HP to systematic form by multiplication of invertible matrix Q (compare to Equation (11)): $\tilde{H} \leftarrow (QHP)_{[k]}$, $\tilde{\mathbf{s}} \leftarrow Q\mathbf{s}$
5: $L_i = \left\{\mathbf{x}_i \mid \mathbf{x}_i = (\mathbf{y}, 0^{k/2}) \colon \mathbf{y} \in \mathcal{B}_{p_1/2}^{k/2}\right\}$ for $i = 1, 3$
6: $L_i = \left\{\mathbf{x}_i \mid \mathbf{x}_i = (0^{k/2}, \mathbf{y}) \colon \mathbf{y} \in \mathcal{B}_{p_1/2}^{k/2}\right\}$ for $i = 2, 4$
7: Compute L_{12}, L_{34} and L_{1234} using nearest neighbor algorithm
8: $L_{12} \leftarrow \{\mathbf{x}_1 + \mathbf{x}_2 \mid (\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2 \wedge \tilde{H}_1(\mathbf{x}_1 + \mathbf{x}_2) \approx_{\omega_{11}} \mathbf{0}\}$
9: $L_{34} \leftarrow \{\mathbf{x}_3 + \mathbf{x}_4 \mid (\mathbf{x}_3, \mathbf{x}_4) \in L_3 \times L_4 \wedge \tilde{H}_1(\mathbf{x}_3 + \mathbf{x}_4) \approx_{\omega_{11}} \tilde{\mathbf{s}}_1\}$
10: $L_{1234} \leftarrow \{\mathbf{x}_{12} + \mathbf{x}_{34} \mid (\mathbf{x}_{12}, \mathbf{x}_{34}) \in L_{12} \times L_{34} \wedge \tilde{H}_2(\mathbf{x}_{12} + \mathbf{x}_{34}) \approx_{\omega_2} \tilde{\mathbf{s}}_2\}$
11: **for** $\mathbf{x} \in L_{1234}$ **do**
12: $\tilde{\mathbf{e}} = (\mathbf{x}, \tilde{H}\mathbf{x} + \tilde{\mathbf{s}})$
13: **if** $\mathrm{wt}(\tilde{\mathbf{e}}) = \omega$ **then**
14: **break**
15: **until** $\mathrm{wt}(\tilde{\mathbf{e}}) = \omega$
16: **return** $P\tilde{\mathbf{e}}$

Hence, as long as we ensure $R \cdot q \geq 1$, we expect at least one representation to survive the restrictions imposed. Even if $R \cdot q < 1$, we can compensate for it by $\frac{1}{R \cdot q}$ randomized constructions of the tree (line 5 to 14 of Algorithm 3).[8]

Lets now turn our focus to the time complexity. We define T_P to be the expected number of random permutations until at least one of them distributes the weight as desired. For each of these T_P permutations we need to apply the Gaussian elimination at a cost of T_G as well as the computation of base lists, three nearest neighbor computations and the weight check of elements of the final list.

Note that in our formulation of the lists they only hold the respective candidates $\mathbf{x}_i$ to keep the notation comprehensible. In a practical application one might also want to store the label $\tilde{H}\mathbf{x}_i$ in L_i for $i = 1, 2, 3$ and respectively $\tilde{H}\mathbf{x}_4 + \tilde{\mathbf{s}}$ in L_4 to avoid their re-computation at later levels. While these labels can be naively computed using matrix vector multiplication, a more sophisticated strategy enumerates the $\mathbf{x}_i$ in the base lists such that $\mathrm{wt}(\mathbf{x}_i + \mathbf{x}_{i+1}) = 2$. Then every label can be computed from the previous one using only two vector additions, yielding a total cost of roughly $\frac{p_1}{2} + 2L_1$ per base list. This is surpassed by the cost for

[8] For example we can randomize by adding a random $\mathbf{r} \in \mathbb{F}_2^{n-k}$ to all labels in lists L_1 and L_3.

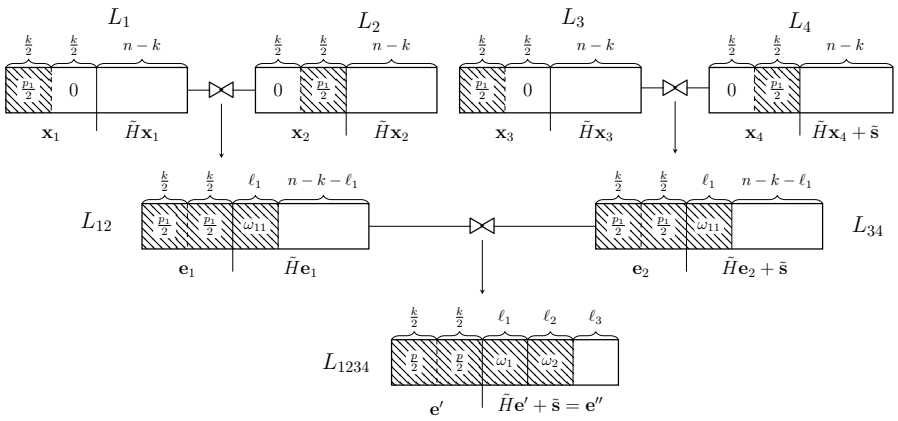

Fig. 1: Graphical representation of the computation tree used by the ISD framework.

the nearest neighbor search on the first level, which is why we neglect this term in the analysis. Let us denote the cost for nearest neighbor search on two lists of size $|L|$, for vectors of length ℓ and weight δ as $\mathcal{N}_{L,\ell,\delta}$. Finally observe, that the computation of L_{1234} comes at a much higher cost than the linear check of the list, which is why we disregard the linear pass through the list in the analysis.

Hence, in total the time complexity becomes

$$T = \underbrace{T_P}_{\text{permutations}} \cdot \Big(\underbrace{T_G}_{\text{Gaussian}} + \underbrace{\max\left(1, (R \cdot q)^{-1}\right)}_{\text{representations}} \cdot \underbrace{\left(2 \cdot \mathcal{N}_{|L_1|,\ell_1,\omega_{11}} + \mathcal{N}_{|L_{12}|,\ell_2,\omega_2}\right)}_{\text{tree computation}} \Big). \tag{15}$$

Note that from a memory point of view one can implement the procedure in a streaming manner rather than storing each list on every level, similar to the ones described in [24, 33]. In this way we only need space for two base lists as well as one intermediate list, since the final list can be checked on-the-fly anyway. Additionally we need to store the matrix, thus we have

$$M = 2 \cdot |L_1| + |L_{12}| + n - k \ .$$

Using the above framework we obtain several ISD algorithms by changing the specific configuration. This includes improved and practical variants of all major ISD improvements (restricted to depth 2) such as MMT, BJMM, May-Ozerov as well as the latest improvement by Both and May.

Let us briefly sketch how to obtain these variants of known ISD algorithms before analyzing them in more detail. Let us start with the MMT/ BJMM algorithm.[9]. Therefore let us instantiate our framework using the Meet-in-the-

[9] Note that we do not differentiate between both algorithms in the following, since the only difference of the BJMM algorithm is the larger choice of the weight of base list elements as $p_1 > p/2$ (and the increase of depth to three, which is not relevant for our practical analysis). Hence, the bare optimization of p_1 determines which algorithm is chosen.

Middle algorithm on both levels to directly obtain a variant of the MMT/ BJMM algorithm using disjoint weight distributions and resolving the shortcomings outlined in the previous section. If we instead choose Meet-in-the-Middle on the first but May-Ozerov nearest neighbor search on the second level, set $\ell_2 = n - k - \ell_1$, $\omega_2 = \omega - p$ and hence $\ell_3 = \omega_3 = 0$ we obtain a variant of the May-Ozerov ISD algorithm using a disjoint weight distribution on the lower level. Observe that the choice of parameters ensures the nearest neighbor search on the final level being performed on all remaining coordinates. And finally if we choose the May-Ozerov nearest neighbor algorithm on both levels with the same choice of $\ell_2, \ell_3, \omega_2, \omega_3$ as for the May-Ozerov variant we obtain the ISD variant of Both and May.

4.1 Concrete Practical Instantiations of the Framework

So let us start the analysis with a variant where we instantiate the nearest neighbor routine by Indyk-Motwani. We credit this variant to Both and May, who first used explicit nearest neighbor search on all levels, by simply calling the variant Both-May in the following.

Remark 4.2 (Balanced weight distribution). As outlined in Section 2, the way we construct the solution only allows to obtain $\mathbf{e}'$ with balanced weight, i.e., vectors having weight $p/2$ on the upper and weight $p/2$ on the lower half of their coordinates. The amount of balanced vectors is a polynomial fraction of all vectors with weight p and hence it is usually disregarded in the theoretical analysis. However, for our practical estimates we account for this. Specifically this influences the amount of representations R as well as the amount of necessary permutations T_P.

Both-May Algorithm. Recall that for the Both-May algorithm we choose $\ell_2 = n - k - \ell_1$, $\omega_2 = \omega - p - \omega_1$ and $\ell_3 = \omega_3 = 0$. Thus the expected amount of iterations until we draw a permutation that distributes the weight according to Equation (13) (under consideration of Remark 4.2) becomes

$$T_P = \frac{\binom{n}{\omega}}{\binom{\ell_2}{\omega_2}\binom{\ell_1}{\omega_1}\binom{k/2}{p/2}^2} = \frac{\binom{n}{\omega}}{\binom{n-k-\ell_1}{\omega-p-\omega_1}\binom{\ell_1}{\omega_1}\binom{k/2}{p/2}^2} \ .$$

Further note, that the number of representations of one balanced vector $\mathbf{e}' \in \mathbb{F}_2^k$ with weight p as a sum of two balanced vectors with weight p_1 is

$$R = \binom{p/2}{p/4}^2 \binom{(k-p)/2}{p_1/2 - p/4}^2 \ .$$

Here the first factor counts the ones contributed to one half of $\mathbf{e}'$ from the first addend. The remaining $p/4$ ones are then contributed by the second addend. The second factor counts the possibilities how the remaining $p1/2 - p/4$ ones in one half can cancel out. Finally since every representation of the lower half can be combined with any representation of the upper half to obtain a valid representation of $\mathbf{e}'$, we square the result.

The probability q of a representation $(\mathbf{e}_1, \mathbf{e}_2)$ of $\mathbf{e}'$ fulfilling the restriction imposed by L_{12} and L_{34} is

$$\begin{aligned} q &:= \Pr\left[\mathrm{wt}(\tilde{H}_1\mathbf{e}_1) = \mathrm{wt}(\tilde{H}_1\mathbf{e}_2 + \tilde{\mathbf{s}}_1) = \omega_{11} \mid \mathbf{e}' = \mathbf{e}_1 + \mathbf{e}_2\right] \\ &= \frac{\binom{\omega_1}{\omega_1/2}\binom{\ell_1-\omega_1}{\omega_{11}-\omega_1/2}}{2^{\ell_1}} \end{aligned}$$

Therefore observe that for a representation $(\mathbf{e}_1, \mathbf{e}_2)$ of $\mathbf{e}'$ it holds that

$$\tilde{H}_1\mathbf{e}_1 + \tilde{H}_1\mathbf{e}_2 + \tilde{\mathbf{s}}_1 = \mathbf{e}_1'', \text{ where } \mathrm{wt}(\mathbf{e}_1'') = \omega_1 \ .$$

Now there are 2^{ℓ_1} different combinations of values for $\tilde{H}_1\mathbf{e}_1, \tilde{H}_1\mathbf{e}_2 + \tilde{\mathbf{s}}_1$ that satisfy the above identity. Out of these pairs $\binom{\omega_1}{\omega_1/2}\binom{\ell_1-\omega_1}{\omega_{11}-\omega_1/2}$ have the correct weight ω_{11}. Now by the randomness of $\tilde{H}_1$ the probability becomes the claimed fraction.

In the base lists we enumerate all vectors of length $k/2$ and weight $p_1/2$, hence it holds

$$|L_1| = \binom{k/2}{p_1/2} \ .$$

The intermediate lists hold all elements of the Cartesian product of two base lists which fulfill the weight restriction on ℓ_1 coordinates, thus

$$|L_{12}| = \frac{|L_1|^2\binom{\ell_1}{\omega_{11}}}{2^{\ell_1}}$$

Eventually the running time $\mathcal{N}_{|L_1|,\ell_1,\omega_{11}}$ for the nearest neighbor routine on the first level and $\mathcal{N}_{|L_{12}|,n-k-\ell_1,\omega-p-\omega_1}$ for the second level are given by Equation (2).

BJMM-DW: BJMM/ MMT with disjoint weight distribution. In comparison to our version of the Both-May algorithm the BJMM-DW algorithm uses MEET-IN-THE-MIDDLE for nearest neighbor search. Thus, we choose $\ell_2 < n-k-\ell_1$ and $\omega_2 < \omega - p - \omega_1$, which yields $\ell_3, \omega_3 > 0$.[10] Accordingly the time complexity for the nearest neighbor search on the first and second level, which are $\mathcal{N}_{|L_1|,\ell_1,\omega_{11}}$ and $\mathcal{N}_{|L_{12}|,\ell_2,\omega_2}$ are now given by Equation (1). Note that the choice of the meet-in-the-middle approach only allows to find elements with balanced distances. Thus, we also need the balanced property on the ℓ_2 and ℓ_1 windows. Hence, the number of permutations and the probability of a representation matching the restrictions change to

$$T_P = \frac{\binom{n}{\omega}}{\binom{n-k-\ell_1-\ell_2}{\omega-\omega_1-\omega_2-p}\binom{\ell_2/2}{\omega_2/2}^2\binom{\ell_1/2}{\omega_1/2}^2\binom{k/2}{p/2}^2} \quad \text{and} \quad q = \frac{\binom{\omega_1/2}{\omega_1/4}^2\binom{\ell_1/2-\omega_1/2}{\omega_{11/2}-\omega_1/4}^2}{2^{\ell_1}} \ .$$

The rest stays as in the analysis of the Both-May variant.

[10] If we would perform the MEET-IN-THE-MIDDLE on the full $n-k-\ell_1$ coordinates as before, the blow-up due to the internal addition of the fixed Hamming weight vectors would be to huge and render this approach inefficient.

4.2 Joint weight distributions

Since in practice ℓ_1 is comparably (to the code length) small and the error weight only sublinear, an optimization of parameters in some cases yields $\omega_{11} = 0$ and hence $\omega_1 = 0$. In these cases we find that a joint weight distribution on the base level, meaning an enumeration of vectors of length $\frac{k+\ell_1}{2}$ rather than $\frac{k}{2}$, as in the original algorithm by Dumer, can yield improvements. Recall that asymptotically the joint weight case is subsumed by the disjoint weight case when using proportional weight on both sides.

However, since our primary focus lies on the concrete hardness of cryptographic parameter sets, which all use a sublinear error weight, we now describe a variant of the framework using a joint weight distribution on the first level. Note that this description is also closer to the original descriptions of the May-Ozerov, BJMM and MMT algorithms, which all use joint weight distributions, the latter even over multiple levels.

First assume that the weight on the solution $\mathbf{e} = (\mathbf{e}', \mathbf{e}''_1, \mathbf{e}''_2, \mathbf{e}''_3) \in \mathbb{F}_2^k \times \mathbb{F}_2^{\ell_1} \times \mathbb{F}_2^{\ell_2} \times \mathbb{F}_2^{\ell_3}$ distributes as

$$\mathrm{wt}((\mathbf{e}', \mathbf{e}''_1)) = p,\ \mathrm{wt}(\mathbf{e}''_2) = \omega_2 \text{ and } \mathrm{wt}(\mathbf{e}''_3) = \omega_3\ . \tag{16}$$

Also we re-randomize the identity part of size ℓ_1 by considering the parity check matrix being of form

$$H = \left(\tilde{H}\ I_{n-k-\ell_1}\right) = \begin{pmatrix} \tilde{H}_1 & \mathbf{0} & \mathbf{0} \\ \tilde{H}_2 & I_{\ell_2} & \mathbf{0} \\ \tilde{H}_3 & \mathbf{0} & I_{\ell_3} \end{pmatrix}\ ,$$

where $\tilde{H} \in \mathbb{F}_2^{(n-k)\times(k+\ell_1)}$ has random structure. This allows us to perform a meet-in-the-middle on $(\mathbf{e}', \mathbf{e}''_1)$ without splitting both parts individually. Therefore we change the definition of base lists to

$$\begin{aligned} L_i &= \left\{\mathbf{x}_i \mid \mathbf{x}_i = (\mathbf{y}, 0^{(k+\ell_1)/2}) \colon \mathbf{y} \in \mathcal{B}_{p_1/2}^{(k+\ell_1)/2}\right\} \text{ for } i = 1, 3 \text{ and} \\ L_i &= \left\{\mathbf{x}_i \mid \mathbf{x}_i = (0^{(k+\ell_1)/2}, \mathbf{y}) \colon \mathbf{y} \in \mathcal{B}_{p_1/2}^{(k+\ell_1)/2}\right\} \text{ for } i = 2, 4\ . \end{aligned}$$

Now we construct the lists L_{12} and L_{34} as

$$\begin{aligned} L_{12} &:= \{\mathbf{x}_1 + \mathbf{x}_2 \mid (\mathbf{x}_1, \mathbf{x}_2) \in L_1 \times L_2 \wedge \tilde{H}_1(\mathbf{x}_1 + \mathbf{x}_2) = \mathbf{0}\} \\ L_{34} &:= \{\mathbf{x}_3 + \mathbf{x}_4 \mid (\mathbf{x}_3, \mathbf{x}_4) \in L_3 \times L_4 \wedge \tilde{H}_1(\mathbf{x}_3 + \mathbf{x}_4) = \tilde{\mathbf{s}}_1\}\ . \end{aligned} \tag{17}$$

Finally list L_{1234} is constructed via nearest neighbor search, as before as

$$L_{1234} \leftarrow \{\mathbf{x}_{12} + \mathbf{x}_{34} \mid (\mathbf{x}_{12}, \mathbf{x}_{34}) \in L_{12} \times L_{34} \wedge \tilde{H}_2(\mathbf{x}_{12} + \mathbf{x}_{34}) \approx_{\omega_2} \tilde{\mathbf{s}}_2\}\ .$$

Adaptation of the analysis. While most of the analysis stays the same as for the general nearest neighbor framework, our adaptations affect some of the details. Precisely the probability q of a representation surviving the imposed restrictions

as well as the cost for the construction of L_{12} and L_{34}. First note that for lists as defined in Equation (17) the probability q is defined as

$$q := \Pr\left[\tilde{H}_1\mathbf{e}_1 = \mathbf{0} \wedge \tilde{H}_1\mathbf{e}_2 = \tilde{\mathbf{s}}_1 \mid (\mathbf{e}', \mathbf{e}_1'') = \mathbf{e}_1 + \mathbf{e}_2\right] = 2^{-\ell_1} \ .$$

Since we already know that $\tilde{H}_1(\mathbf{e}', \mathbf{e}_1'') = \tilde{\mathbf{s}}_1$ by randomness of $\tilde{H}_1$ we have $q = \Pr\left[\tilde{H}_1\mathbf{e}_1 = \mathbf{0}\right] = 2^{-\ell_1}$. Now observe that the construction of L_{12} (resp. L_{34}) can be done in time $|L_1| + |L_2| + |L_{12}|$ as we only need to check for equality. Thus, the runtime can still be expressed via Equation (15), where $\mathcal{N}_{|L_1|,\ell_1,\omega_{11}} := |L_1| + |L_2| + |L_{12}|$. Next we give two instantiations with joint weight distribution on the base level. The first is a variant of the BJMM algorithm using again Meet-in-the-Middle for the construction of L_{1234}, while the second uses Indyk-Motwani instead and can be seen as a variant of the May-Ozerov algorithm.

BJMM-p-dw: BJMM/ MMT algorithm with partially disjoint weight distribution. The expected amount of permutations until we may expect a weight distribution as given in Equation (16) under consideration of the balanced property (see Remark 4.2) is

$$T_P = \frac{\binom{n}{\omega}}{\binom{n-k-\ell_1-\ell_2}{\omega-\omega_1-\omega_2-p}\binom{(k+\ell_1)/2}{p/2}^2\binom{\ell_2/2}{\omega_2/2}^2}$$

Note that the balanced property on the ℓ_2 windows stems from the fact that the BJMM algorithm uses Meet-in-the-Middle for the construction of L_{1234}. The base lists are now of increased size $|L_1| = \binom{(k+\ell_1)/2}{p_1/2}$ while $|L_{12}| = |L_{34}| = |L_1|^2/2^{\ell_1}$ since we perform an exact matching on ℓ_1 coordinates. On the upside we now have an increased amount of representations

$$R = \binom{p/2}{p/4}^2\binom{(k+\ell_1-p)/2)}{p_1/2-p/4}^2 \ .$$

The cost for the computation of L_{1234}, namely $\mathcal{N}_{|L_{12}|,\ell_2,\omega_2}$, is given by Equation (1).

May-Ozerov The essential difference to the BJMM-p-dw lies in the usage of the Indyk-Motwani for the computation of L_{1234}. Also this variant chooses $\ell_2 = n - k - \ell_1$ and $\omega_2 = \omega - p$, which implies $\ell_3 = \omega_3 = 0$. Then the complexity of the final list computation $\mathcal{N}_{|L_{12}|,\ell_2,\omega_2}$ is given by Equation (2). The rest stays as in the analysis of the BJMM-p-dw.

For completeness and comparisons we also state the running time of the original BJMM algorithm (in depth two). It uses a joint weight distribution over both levels, hence the vectors are enumerated on $(k + \ell_1 + \ell_2)/2$ coordinates in the base lists. This results in the final construction of list L_{1234} coming at a cost of $|L_{12}| + |L_{34}| + |L_{1234}|$. Referring to Equation (15) this means we have $\mathcal{N}_{|L_{12}|,\ell_2,\omega_2} = |L_{12}| + |L_{34}| + |L_{1234}|$

BJMM: original BJMM algorithm. Let $\ell = \ell_1 + \ell_2$. The base list size of the BJMM algorithm is $|L_1| = \binom{(k+\ell)/2}{p_1/2}$. The matching is then performed again on ℓ_1 coordinates, which results in $|L_{12}| = |L_{34}| = |L_1|^2/2^{\ell_1}$. The amount of representations is

$$R = \binom{p/2}{p/4}^2 \binom{(k+\ell-p)/2}{p_1/2 - p/4}^2.$$

Now the construction of list L_{1234} is performed as a matching of L_{12} and L_{34} on $\ell - \ell_1 = \ell_2$ coordinates, thus we have $|L_{1234}| = |L_{12}|^2/2^{\ell_2}$. The rest stays as in the BJMM-P-DW algorithm.

5 Estimator

In this section we present our results on the bit security estimates for the suggested parameters of code based cryptographic submissions to the NIST PQC standardisation process, namely McEliece, BIKE and HQC.

A cautious note on concrete hardness estimates. Concrete hardness estimates often give the impression of being highly accurate, not least because they are usually given up to the second or even third decimal place. In particular, following recent discussions (in the NIST PQC forum [31]), we want to emphasize that these concrete security estimates should always be taken with care. They heavily rely on implementation details, targeted platforms, possible hardware accelerations and many more factors. Thus, many assumptions must be made to obtain these estimates. The derived numbers should therefore be understood as *indicative* rather than precise. Following this line of thought we give our estimates rounded to the nearest integer and admit that they may inherit an inaccuracy of a few bits.

Before we discuss the security estimations let us briefly address some methodology aspects. All advanced algorithms we consider require an exponential amount of memory, which certainly slows down computations compared to memory-free algorithms like Prange. In [3] it was suggested to use a logarithmic memory access model, which accounts for the need of memory with an additive $\log\log \texttt{memory}$ in the bit security estimate, where `memory` is the total memory consumed by the algorithm. We follow this suggestion and consider besides the more conservative constant memory access cost also this logarithmic model. Additionally, we consider a cube-root memory access cost, which results in an additive $\log \sqrt[3]{\texttt{memory}}$ in the respective algorithms bit complexity, which was recently suggested by Perlner [25].

Note that our estimator software also allows for a square-root access cost model as well as user defined cost functions. However, we believe that the constant, logarithmic and cube-root models already give a good overview.

The NIST defines five security categories, where most submissions focus on parameters for the categories one, three and five. A parameter set for a proposal is said to match the security level of category one, three or five if the scheme instantiated with the corresponding parameters is as at least as hard to break as

	Category	n	k	ω
McEliece	1	3488	2720	64
	3	4608	3360	96
	5	6688	5024	128
	5	6960	5413	119
	5	8192	6528	128
BIKE (message)	1	24646	12323	134
	3	49318	24659	199
	5	81946	40973	264
BIKE (key)	1	24646	12323	142
	3	49318	24659	206
	5	81946	40973	274
HQC	1	35338	17669	132
	3	71702	35851	200
	5	115274	57637	262

Table 1: Parameter sets suggested by NIST PQC proposals.

AES-128, AES-192 or AES-256 respectively. In Table 1 we list all corresponding parameters of each submission and their respective security category.

Remark 5.1 (Up-to-date Estimates). Even though, we computed the presented estimates with the utmost care, we would like to encourage the reader to use our *Syndrome Decoding Estimator* rather than only relying on the estimates given in this paper. This is because the tool offers a wide variety of customization to make sure the estimates address the right setting. Also, we are constantly extending and improving the *Syndrome Decoding Estimator* such that the results obtained might slightly vary from the tables presented here.

Let us start with the Classic McEliece submission. Table 2 shows the bit complexity estimates for all suggested parameter sets. Besides the ISD variants obtained as instantiations of our framework we included the estimates for Prange and Stern for comparisons. It can be observed that the time complexity estimations for all advanced ISD algorithms, namely Both-May, May-Ozerov, BJMM, BJMM-dw and BJMM-p-dw are comparable, where the variants bring slight advantages over BJMM. However, the use of explicit nearest neighbor search and disjoint weight distributions pays off when turning the focus to the memory consumption. For the proposed parameter sets, our new BJMM variants for instance allow to decrease the memory consumption of plain BJMM by up to 30 bits. This partly stems form the advantage of not enumerating unnecessary vectors in the base lists as outlined in Section 3. Recall that this reduced memory consumption eventually also results in practical run time improvements when considering different memory access models.

	Category 1 ($n = 3488$)		Category 3 ($n = 4608$)		Category 5 (n=6688)		Category 5 ($n = 6960$)		Category 5 ($n = 8192$)	
	T	M	T	M	T	M	T	M	T	M
Prange	173	22	217	23	296	24	297	24	334	24
Stern	151	50	193	60	268	80	268	90	303	109
Both-May	143	88	182	101	250	136	249	137	281	141
May-Ozerov	141	89	180	113	246	165	246	160	276	194
BJMM	142	97	183	121	248	160	248	163	278	189
BJMM-p-dw	143	86	183	100	249	160	248	161	279	166
BJMM-dw	144	97	183	100	250	130	250	160	282	164
$M \leq 60$	145	60	187	60	262	58	263	60	298	59
$M \leq 80$	143	74	183	77	258	76	258	74	293	77
$\log M$ access	147	89	187	113	253	165	253	160	283	194
$\sqrt[3]{M}$ access	156	25	199	26	275	36	276	36	312	47

Table 2: Bit security estimates for the suggested parameter sets of the Classic McEliece scheme.

Additionally we provide in Table 2 bit-security estimates where the available memory is limited to 60 and 80 bits (still in the constant memory access model). Under this memory restrictions the May-Ozerov algorithm performs best by entirely providing the best estimates for those regimes. This is an effect of the use of joined weight distributions on the lower level as well as memory-efficient nearest neighbor search in form of Indyk-Motwani.

Also we state the best complexities obtained when considering the logarithmic and cube-root memory access model.[11] For the logarithmic model it can be observed that the optimization suggests the same parameters as in the constant model. This results in all bit complexities being increased by a logarithm of the memory usage in the constant setting. Contrary, the optimization in the cube-root model avoids the use of memory almost completely, yielding significantly higher bit complexities.

We also used our estimator to approximate the hardness of an already solved McEliece instance reported online at `decodingchallenge.org` [2] and an instance that was attacked by Bernstein, Lange and Peters in [6]. Recently, Esser, May and Zweydinger reported the solution to an instance with parameters $(n = 1284, k = 1028, w = 24)$ [2], for this instance our estimator yields a bit complexity of 65 bit. For the instance $(n = 1024, k = 525, w = 50)$ attacked in [6] by Bernstein et al. we find a slightly lower bit complexity of 64 bit. Note that while these numbers might occur high, usually attacks are performed with a register-width of 64 bit. Thus, the actual operation count is reduced by six bit, yielding operation counts

[11] Note that we take the number of necessary vector space elements that need to be stored as a measure to derive the penalty.

		Category 1 ($n = 24646$)		Category 3 ($n = 49318$)		Category 5 ($n = 81946$)	
		T	M	T	M	T	M
message security	PRANGE	167	28	235	30	301	32
	STERN	146	40	211	43	277	45
	BOTH-MAY	147	38	212	41	276	63
	MAY-OZEROV	146	55	211	57	276	61
	BJMM	147	38	211	59	277	63
	BJMM-P-DW	147	37	211	56	276	61
	BJMM-DW	147	45	211	56	277	43
	$\log M$ access	150	31	215	33	281	34
	$\sqrt[3]{M}$ access	152	30	217	32	283	33
key security	PRANGE	169	28	234	30	304	32
	STERN	147	40	211	43	279	45
	BOTH-MAY	148	38	211	60	278	63
	MAY-OZEROV	147	55	210	57	278	61
	BJMM	147	54	211	59	279	63
	BJMM-P-DW	147	55	211	56	278	61
	BJMM-DW	147	55	211	56	279	43
	$\log M$ access	151	31	215	33	283	34
	$\sqrt[3]{M}$ access	153	30	217	32	285	33

Table 3: Bit security estimates for the suggested parameter sets of the BIKE scheme.

of 59 and 58 bits for those instances. These estimates seem to be coherent with the reported computational efforts made to solve those instances.

Next we give the security estimates for the BIKE scheme. Note that BIKE uses a quasi-cylic code allowing for polynomial speedups, which need to be considered in the security estimates.

This is also the reason why we have to distinguish the message and key security. Obtaining the message from a BIKE ciphertext requires the attacker to solve a syndrome decoding problem for a syndrome usually not equal to the zero vector. Opposed to that, attacking the secret key requires finding a low weight codeword or equivalently solving the syndrome decoding problem, where the syndrome is the zero vector. For these two cases different speed-ups can be obtained due to the cyclic structure.

In terms of message security, the cyclicity allows to derive $n-k = k$ syndrome decoding instances from an initial input instance out of which a single one has to be decoded to break the security of the system. This variant is known as decoding one out of many (DOOM). It has been shown in [28] that Stern's algorithm can be sped up in this setting by a factor of roughly $\sqrt{k}$. Even though, it has not been studied how to obtain this speed-up for advanced ISD algorithms, such as

	Category 1 ($n = 35338$)		Category 3 ($n = 71702$)		Category 5 ($n = 115274$)	
	T	M	T	M	T	M
Prange	166	29	237	31	300	33
Stern	145	41	213	44	276	46
Both-May	146	39	214	42	276	39
May-Ozerov	145	39	214	42	276	44
BJMM	146	39	214	42	276	44
BJMM-p-dw	146	39	214	42	276	43
BJMM-dw	146	39	214	42	276	40
$\log M$ access	150	32	218	34	280	36
$\sqrt[3]{M}$ access	151	31	220	33	282	35

Table 4: Bit security estimates for the suggested parameter sets of the HQC scheme.

BJMM, it is commonly assumed to be obtainable similar to the case of Stern's algorithm. Hence, we also deducted $\frac{\log k}{2}$ from all our bit security estimates.

Considering key security the quasi cyclic code contains all k cyclic shifts of the searched codeword. Hence, without any adaptations the probability of choosing a permutation that distributes the weight as desired is enhanced by a factor of k. Thus, in this case we subtract $\log(k)$ from our bit security estimates.

Table 3 states the bit security estimates for the BIKE scheme. Note that BIKE in comparison to McEliece uses a decreased error weight of $\omega = \mathcal{O}(\sqrt{n})$ rather than $\omega = \mathcal{O}(\frac{n}{\log n})$. This reduced weight lets the advantage of enumeration based algorithms deteriorate, with the result that advanced algorithms only offer a slight advantage over basic Stern. However, when considering the logarithmic or cube-root model still our May-Ozerov variant provides the best complexities. For the BIKE setting we observe that already for a logarithmic penalty the optimization suggests to use low-memory configurations.

Eventually we state in Table 4 our bit security estimates for the HQC scheme. Similar to the BIKE scheme HQC uses a cyclic structure allowing for a $\sqrt{k}$ speedup. Also HQC builds on the same asymptotically small error weight of $\omega = \mathcal{O}(\sqrt{n})$, but in comparison to BIKE uses even smaller constants. Thus, as the estimates show, the advantage of more involved procedures vanishes almost completely. Here most variants degenerate to a version of Stern, by choosing all intermediate weights equal to zero. Nevertheless, when considering memory access costs again our May-Ozerov algorithm yields the best time complexity.

The biggest solved instance reported to `decodingchallenge.org` in the quasi-cyclic setting has parameters $(n = 2918, k = 1459, w = 54)$ [2]. We estimate for this instance a bit complexity of 64 bit using our estimator tool, which corresponds to roughly 2^{58} necessary operations on a 64-bit architecture.

Interpretation of the security estimates. We observe that most bit security estimates match the required classical gate counts of 143, 207 and 272, corresponding to breaking AES-128, AES-192 and AES-256 according to NIST, surprisingly well. Note that all submissions use the asymptotic complexity estimate of basic Prange as foundation for parameter selection. Hence, the exact match comes from the fact that the improvement due to advanced algorithms corresponds quite exactly to Prange's polynomial factors.

The McEliece submission team admits that advanced ISD algorithms yield a lower complexity count then required in the constant memory access model [10], which is confirmed by our estimates. However, they argue that the memory access costs faced in the real world will make up for the difference. Our estimates support this claim, when imposing a cube-root memory access cost.

Only the category three parameter set of the Classic McEliece scheme seems to fail the given security level slightly. Here, even when imposing a cube-root memory access cost it still deviates from the needed 207 bits by eight bits.

For BIKE and HQC the consideration of different memory access cost models is less significant. This is because already in the constant memory access setting the parameters do not allow advanced ISD techniques to leverage the use of memory. However, both schemes already match their security levels in the more conservative constant cost model. In the more realistic setting with logarithmic or cub-root access cost both schemes' parameter sets seem to inherit a slight margin of five to fifteen bits.

5.1 Quantum Security

The metric for quantum security provided by NIST is based on a $\texttt{maxdepth} \in \{40, 64, 96\}$ constraint, defining the maximum allowed depth of a quantum circuit used to attack the corresponding instantiation. Here the `maxdepth` constraint accounts for the difficulty of constructing large scale quantum computers.

A parameter set is said to match the quantum security definition of category one, three or five, if it is at least as hard to break as AES-128, AES-192 or AES-256 quantumly. Furthermore NIST defines the quantum security of AES as

- AES-128 corresponds to $2^{170-\texttt{maxdepth}}$ quantum gates,
- AES-192 corresponds to $2^{233-\texttt{maxdepth}}$ quantum gates,
- AES-256 corresponds to $2^{298-\texttt{maxdepth}}$ quantum gates.

In terms of quantum attacks Bernstein showed, that the search for a correct permutation in Prange's algorithm can be asymptotically accelerated by a square-root gain using a Grover search [5]. There has also been some research on how to quantize advanced ISD algorithms [17, 18]. While resulting in slightly better asymptotic complexities than Bernstein's quantum Prange, we agree with the BIKE submission [1], that these procedures inherit huge overheads making them for practical security estimates irrelevant.

In the following we analyze the quantum security of the proposed parameter sets based on Bernstein's quantum Prange in a very optimistic way, disregarding any overhead caused by a real implementation. Nevertheless, even with our analysis, which disregards a lot of the actual costs, we are able to show that

all parameter sets still match the quantum security definition of the respective categories.

Quantum Prange under depth constraint. Let the Gaussian elimination circuit require a depth of $D_E = (n-k)^{\omega_M}$, where $\omega_M \in [\![2, 3]\!]$. Now recall that the probability of drawing a random permutation that distributes the weight as desired (see Equation (4)) is

$$q := \frac{\binom{n-k}{\omega}}{\binom{n}{\omega}} \ .$$

Hence, leveraging Grover search, finding a good permutation with one application of a single quantum circuit would require a depth of

$$D = \mathcal{O}(D_E \sqrt{q^{-1}}) \ .$$

Since we are limited in depth, following Zalka [34] we need to partition the search space in enough pieces such that the circuit performing a search on each partition does not exceed the depth constraint. Then the circuit has to be reapplied for each partition. Separating the search space in N equally large partitions results in a necessary quantum circuit depth of

$$D_N = \mathcal{O}(D_E \sqrt{(qN)^{-1}}) \ ,$$

for performing the search on a single partition. Now setting $D_N = 2^{\texttt{maxdepth}}$ results in

$$N = \frac{(D_E)^2 \cdot q^{-1}}{2^{2 \cdot \texttt{maxdepth}}} \ .$$

Finally this gives a quantum circuit of depth $2^{\texttt{maxdepth}}$ which has to be reapplied N times to find the solution. Hence, the total time complexity becomes

$$T_Q = N \cdot 2^{\texttt{maxdepth}} = \frac{(D_E)^2}{q \cdot 2^{\texttt{maxdepth}}} \ .$$

In Table 5 we present the quantum bit security estimates for the NIST PQC schemes. Note that we do not state the security for every category and `maxdepth` combination. Rather we just state the difference of $\log T_Q$ and the quantum bit security of AES for the corresponding category, which is given in the above listing. Observe that this difference does not depend on the `maxdepth` constraint anymore. The table can now be read as *the quantum bit security of breaking the corresponding scheme is x bit higher than required*, where x is the value given in the column "quantum security margin". For obtaining these estimates we used the more than optimistic matrix multiplication constant of $\omega_M = 2.5$.

The estimates confirm our prior finding, that the McEliece parameter set for category three inherits a way smaller security margin compared to the other proposed parameter sets.

Scheme	Category	n	quantum security margin
McEliece	1	3488	21
	3	4608	3
	5	6688	18
	5	6960	18
	5	8192	56
BIKE (message)	1	24646	41
	3	49318	47
	5	81946	53
BIKE (key)	1	24646	32
	3	49318	40
	5	81946	43
HQC	1	35338	33
	3	71702	43
	5	115274	44

Table 5: Quantum bit security margin of the corresponding schemes in comparison to breaking AES quantumly.

References

1. Aragon, N., Barreto, P., Bettaieb, S., Bidoux, L., Blazy, O., Deneuville, J.C., Gaborit, P., Gueron, S., Guneysu, T., Melchor, C.A., et al.: BIKE: bit flipping key encapsulation (2020)
2. Aragon, N., Lavauzelle, J., Lequesne, M.: decodingchallenge.org (2019), http://decodingchallenge.org
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)
4. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: Pointcheval, D., Johansson, T. (eds.) Advances in Cryptology – EUROCRYPT 2012. Lecture Notes in Computer Science, vol. 7237, pp. 520–536. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012). https://doi.org/10.1007/978-3-642-29011-4_31
5. Bernstein, D.J.: Grover vs. McEliece. In: International Workshop on Post-Quantum Cryptography. pp. 73–80. Springer (2010)
6. Bernstein, D.J., Lange, T., Peters, C.: Attacking and defending the mceliece cryptosystem. In: International Workshop on Post-Quantum Cryptography. pp. 31–46. Springer (2008)
7. Bernstein, D.J., Lange, T., Peters, C.: Smaller decoding exponents: Ball-collision decoding. In: Rogaway, P. (ed.) Advances in Cryptology – CRYPTO 2011. Lecture Notes in Computer Science, vol. 6841, pp. 743–760. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2011). https://doi.org/10.1007/978-3-642-22792-9_42

8. Both, L., May, A.: Decoding linear codes with high error rate and its impact for lpn security. In: International Conference on Post-Quantum Cryptography. pp. 25–46. Springer (2018)
9. Canteaut, A., Chabaud, F.: A new algorithm for finding minimum-weight words in a linear code: Application to mceliece's cryptosystem and to narrow-sense bch codes of length 511. IEEE Transactions on Information Theory **44**(1), 367–378 (1998)
10. Chou, T., Cid, C., UiB, S., Gilcher, J., Lange, T., Maram, V., Misoczki, R., Niederhagen, R., Paterson, K.G., Persichetti, E., et al.: Classic McEliece: conservative code-based cryptography 10 october 2020 (2020)
11. Dumer, I.: On minimum distance decoding of linear codes. In: Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory. pp. 50–52 (1991)
12. Esser, A., Kübler, R., May, A.: LPN decoded. In: Katz, J., Shacham, H. (eds.) Advances in Cryptology – CRYPTO 2017, Part II. Lecture Notes in Computer Science, vol. 10402, pp. 486–514. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63715-0_17
13. Esser, A., Kübler, R., Zweydinger, F.: A faster algorithm for finding closest pairs in hamming metric. arXiv preprint arXiv:2102.02597 (2021)
14. Finiasz, M., Sendrier, N.: Security bounds for the design of code-based cryptosystems. In: Matsui, M. (ed.) Advances in Cryptology – ASIACRYPT 2009. Lecture Notes in Computer Science, vol. 5912, pp. 88–105. Springer, Heidelberg, Germany, Tokyo, Japan (Dec 6–10, 2009). https://doi.org/10.1007/978-3-642-10366-7_6
15. Hamdaoui, Y., Sendrier, N.: A non asymptotic analysis of information set decoding. IACR Cryptol. ePrint Arch. **2013**, 162 (2013)
16. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the thirtieth annual ACM symposium on Theory of computing. pp. 604–613 (1998)
17. Kachigar, G., Tillich, J.P.: Quantum information set decoding algorithms. In: International Workshop on Post-Quantum Cryptography. pp. 69–89. Springer (2017)
18. Kirshanova, E.: Improved quantum information set decoding. In: International Conference on Post-Quantum Cryptography. pp. 507–527. Springer (2018)
19. Kirshanova, E., Laarhoven, T.: Lower bounds on lattice sieving and information set decoding. To appear at CRYPTO 2021 (2021)
20. 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. Lecture Notes in Computer Science, vol. 330, pp. 275–280. Springer, Heidelberg, Germany, Davos, Switzerland (May 25–27, 1988). https://doi.org/10.1007/3-540-45961-8_25
21. May, A., Meurer, A., Thomae, E.: Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In: Lee, D.H., Wang, X. (eds.) Advances in Cryptology – ASIACRYPT 2011. Lecture Notes in Computer Science, vol. 7073, pp. 107–124. Springer, Heidelberg, Germany, Seoul, South Korea (Dec 4–8, 2011). https://doi.org/10.1007/978-3-642-25385-0_6
22. May, A., Ozerov, I.: On computing nearest neighbors with applications to decoding of binary linear codes. In: Oswald, E., Fischlin, M. (eds.) Advances in Cryptology – EUROCRYPT 2015, Part I. Lecture Notes in Computer Science, vol. 9056, pp. 203–228. Springer, Heidelberg, Germany, Sofia, Bulgaria (Apr 26–30, 2015). https://doi.org/10.1007/978-3-662-46800-5_9
23. 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) (2020)

24. Naya-Plasencia, M., Schrottenloher, A.: Optimal merging in quantum k-xor and k-sum algorithms. In: EUROCRYPT 2020-39th Annual International Conference on the Theory and Applications of Cryptographic (2020)
25. Perlner, R.: pqc-forum: Round 3 official comment: Classic mceliece (2021), available at: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/EiwxGnfQgec/m/xBky_FKFDgAJ
26. Peters, C.: Information-set decoding for linear codes over f q. In: International Workshop on Post-Quantum Cryptography. pp. 81–94. Springer (2010)
27. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
28. Sendrier, N.: Decoding one out of many. In: International Workshop on Post-Quantum Cryptography. pp. 51–67. Springer (2011)
29. Stern, J.: A method for finding codewords of small weight. In: International Colloquium on Coding Theory and Applications. pp. 106–113. Springer (1988)
30. Torres, R.C., Sendrier, N.: Analysis of information set decoding for a sub-linear error weight. In: Post-Quantum Cryptography. pp. 144–161. Springer (2016)
31. Various: pqc-forum: Round 3 official comment: Classic mceliece (2021), available at: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/EiwxGnfQgec
32. Various: pqc-forum: Security strength categories for code based crypto (and trying out crypto stack exchange) (2021), available at: https://groups.google.com/a/list.nist.gov/g/pqc-forum/c/6XbG66gI7v0
33. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) Advances in Cryptology – CRYPTO 2002. Lecture Notes in Computer Science, vol. 2442, pp. 288–303. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2002). https://doi.org/10.1007/3-540-45708-9_19
34. Zalka, C.: Grover's quantum searching algorithm is optimal. Physical Review A **60**(4), 2746 (1999)

A Remark on the correctness of recently obtained estimates

The recent analysis of the MMT and BJMM algorithm conducted in [3] contains a subtle but essential flaw leading to significant underestimation of the respective algorithmic costs. More precisely the authors suggest for the MMT algorithm to choose the ℓ_2 value (page 18 of [3]) as $\log \binom{(k+l)/2}{p/4}$, where this value corresponds to the amount of bits on which the matching in the first level is performed. But since there exist only $\binom{p}{p/2}$ representations of the solution, this results in an expected amount of

$$\frac{\binom{(k+l)/2}{p/4}}{\binom{p}{p/2}}$$

necessary randomized iterations of the algorithm to find a solution for any permutation distributing the weight as desired. However, the authors disregard this factor in the complexity analysis, leading to the mentioned underestimation.

Contrary, for the BJMM algorithm the authors choose the values for ℓ_1 and ℓ_2 correctly as the logarithm of the existing representations. However, since they use a BJMM algorithm with a search tree of depth three, they need to account for the first stage matching in the second level by only constraining on $\ell_1 - \ell_2$ instead of ℓ_1 bits.