McEliece needs a Break – Solving McEliece-1284 and Quasi-Cyclic-2918 with Modern ISD

Andre Esser[1], Alexander May[2*], and Floyd Zweydinger[2†]

[1] Cryptography Research Center, Technology Innovation Institute, UAE
andre.esser@tii.ae
[2] Ruhr University Bochum, Germany
{alex.may,floyd.zweydinger}@rub.de

Abstract. With the recent shift to post-quantum algorithms it becomes increasingly important to provide precise bit-security estimates for code-based cryptography such as McEliece and quasi-cyclic schemes like BIKE and HQC. While there has been significant progress on information set decoding (ISD) algorithms within the last decade, it is still unclear to which extent this affects current cryptographic security estimates.
We provide the first concrete implementations for representation-based ISD, such as May-Meurer-Thomae (MMT) or Becker-Joux-May-Meurer (BJMM), that are parameter-optimized for the McEliece and quasi-cyclic setting. Although MMT and BJMM consume more memory than naive ISD algorithms like Prange, we demonstrate that these algorithms lead to significant speedups for practical cryptanalysis on medium-sized instances (around 60 bit). More concretely, we provide data for the record computations of McEliece-1223 and McEliece-1284 (old record: 1161), and for the quasi-cyclic setting up to code length 2918 (before: 1938).
Based on our record computations we extrapolate to the bit-security level of the proposed BIKE, HQC and McEliece parameters in NIST's standardization process. For BIKE/HQC, we also show how to transfer the Decoding-One-Out-of-Many (DOOM) technique to MMT/BJMM. Although we achieve significant DOOM speedups, our estimates confirm the bit-security levels of BIKE and HQC.
For the proposed McEliece round-3 192 bit and two out of three 256 bit parameter sets, however, our extrapolation indicates a security level overestimate by roughly 20 and 10 bits, respectively, i.e., the high-security McEliece instantiations may be a bit less secure than desired.

Keywords: MMT/BJMM Decoding, Representation Technique, McEliece

1 Introduction

For building trust in cryptographic instantiations it is of utmost importance to provide a certain level of real-world cryptanalysis effort. Code-based cryptography

* Funded by DFG under Germany's Excellence Strategy - EXC 2092 CASA - 390781972.
† Funded by BMBF under Industrial Blockchain – iBlockchain

is usually build on the difficulty of correcting errors in binary linear codes. Let C be a binary linear code of length n and dimension k, i.e., C is a k-dimensional subspace of $\mathbb{F}_2^n$. We denote by $H \in \mathbb{F}_2^{(n-k)\times n}$ a parity-check matrix of C, thus we have $H\mathbf{c} = \mathbf{0}$ for all $\mathbf{c} \in C$.

Let $\mathbf{x} = \mathbf{c} + \mathbf{e}$ be an erroneous codeword with error $\mathbf{e}$ of small known Hamming weight $\omega = \mathrm{wt}(\mathbf{e})$. Let $\mathbf{s} := H\mathbf{x} = H\mathbf{e}$ denote the syndrome of $\mathbf{x}$. Then decoding $\mathbf{x}$ is equivalent to the recovery of the weight-ω error vector from $H\mathbf{e} = \mathbf{s}$.

Permutation-Dominated ISD – Prange. Let $P \in \mathbb{F}_2^{n\times n}$ be a permutation matrix. Then $(HP)(P^{-1}\mathbf{e}) = \bar{H}\bar{\mathbf{e}} = \mathbf{s}$ is another weight-ω decoding instance with permutated solution $\bar{\mathbf{e}} = P^{-1}\mathbf{e}$.

Assume that $\bar{\mathbf{e}} = (\mathbf{e}_1, \mathbf{e}_2)$ with $\mathbf{e}_2 = 0^k$. An application of Gaussian elimination $G \in \mathbb{F}_2^{(n-k)\times(n-k)}$ on the first $n-k$ columns of $\bar{H}$ yields

$$G\bar{H}\bar{\mathbf{e}} = (I_{n-k}H')\bar{\mathbf{e}} = \mathbf{e}_1 + H'\mathbf{e}_2 = \mathbf{e}_1 = G\mathbf{s}. \tag{1}$$

Thus, from $\mathrm{wt}(G\mathbf{s}) = \omega$ we conclude that $\bar{\mathbf{e}} = (G\mathbf{s}, 0^k)$ and $\mathbf{e} = P\bar{\mathbf{e}}$.

In summary, if we apply the correct permutation P that sends all weight ω to the first $n-k$ coordinates, then we decode correctly in polynomial time. This is why the first $n-k$ coordinates are called an *information set*, and the above algorithm is called *information set decoding* (ISD). This ISD algorithm, due to Prange [18], is the main tool for estimating the security of code-based cryptography such as McEliece and BIKE/HQC.

We would like to stress that the complexity of Prange's algorithm is mainly dominated by finding a proper permutation P, which takes super-polynomial time for cryptographic instances. All other steps of the algorithm are polynomial time. This is why we call Prange a *permutation-dominated* ISD algorithm. A permutation-dominated ISD performs especially well for small weight errors $\mathbf{e}$ and large co-dimension $n-k$. More precisely, we have to find a permutation P that sends all weight ω to the size-$(n-k)$ information set, which happens with probability

$$\Pr[P \text{ good}] = \frac{\binom{n-k}{\omega}}{\binom{n}{\omega}} = \frac{(n-k)(n-k-1)\dots(n-k-\omega+1)}{n(n-1)\dots(n-\omega+1)}.$$

Let $\omega = o(n)$, and let us denote C's rate by $R = \frac{k}{n}$. Then Prange's permutation-based ISD takes up to polynomial factors expected running time

$$T = \frac{1}{\Pr[P \text{ good}]} \approx \left(\frac{1}{1-R}\right)^{\omega}. \tag{2}$$

Modern Enumeration-Dominated ISD – MMT/BJMM. The core idea of all ISD improvements since Prange's algorithm is to allow for some weight $p > 0$ outside the information set. Thus we allow in Equation (1) that $\mathrm{wt}(\mathbf{e}_2) = p$, and have to *enumerate* $H'\mathbf{e}_2$. However, the cost of enumerating $H'\mathbf{e}_2$ may be well compensated by the larger success probability of a good permutation

$$\Pr[P \text{ good}] = \frac{\binom{n-k}{\omega-p}\binom{k}{p}}{\binom{n}{\omega}}.$$

Indeed, modern ISD algorithms like MMT [15] and BJMM [6] use the *representation technique* to heavily speed up enumeration. In the large weight regime $\omega = \Theta(n)$, parameter optimization of MMT/BJMM yields that these ISD algorithm do not only balance the cost of permutation and enumeration, but their enumeration is so efficient that it eventually almost completely dominates their runtime. This is why we call these algorithms *enumeration-dominated* ISD.

The large error regime $\omega = \Theta(n)$ is beneficial for MMT/BJMM, since for large-weight errors $\mathbf{e}$ it becomes hard to send all weight to the information set, and additionally a large-weight $\mathbf{e}$ introduces a large number of representations. From a cryptographic perspective however it remains unclear if MMT/BJMM also offer speedups for concrete cryptographic instances of interest.

Main question: How much improve modern enumeration-based ISD algorithms cryptanalysis of code-based crypto in practice (if at all)?

What makes this question especially hard to answer is that as opposed to permutation-based ISD, all enumeration-based ISD algorithms require a significant amount of memory. Thus, even if enumeration provides significant speedups it is unclear if it can compensate for the introduced memory access costs. As a consequence, the discussion of enumeration-based ISD in the NIST standardization process of McEliece already led to controversial debates [22, 23]. We would like to stress that up to our work, all decoding records on decodingchallenge.org have been achieved either using Prange's permutation-based ISD, or Dumer's first generation enumeration-based ISD [10].

In the asymptotic setting, Canto-Torres and Sendrier [21] showed that all enumeration-dominated ISD approaches offer in the small-weight setting $\omega = o(n)$ only a speedup from T in Equation (2) to $T^{1-o(1)}$, i.e., the speedup asymptotically vanishes. While this is good news for the overall soundness of our cryptographic constructions, it tells us very little about the concrete hardness of their instantiations.

Recently, Esser and Bellini [11] pursued a more practice-oriented approach by providing a concrete *code estimator*, analogous to the successfully applied *lattice estimators* [2]. Their code estimator also serves us as a basis for optimizing our ISD implementations. However, such an estimator certainly fails to model realistic memory access costs.

1.1 Our Contributions

Fast enumeration-dominated ISD implementation. We provide the first efficient, freely available implementation of MMT/BJMM, i.e., a representation-based enumeration-dominated ISD. Our implementation uses depth 2 search-trees, which seems to provide best results for the cryptographic weight regime. For the

cryptographic instances that we attack we used weight $p = 4$ for McEliece with code length up to 1284, and $p = 3$ for BIKE/HQC. However, our benchmarking predicts that McEliece with code length larger than 1350 should be attacked with significantly larger weight $p = 8$. Our code is publicly available on GitHub.[3]

In comparison to other available implementations of (first-generation) enumeration-based ISD algorithms, our implementation performs significantly faster. Our experimental results demonstrate that in cryptanalytic practice even moderately small instances of McEliece can be attacked faster using modern enumeration-based ISD.

So far, our efforts to additionally speed up our enumeration-based ISD implementations with localilty-sensitive hashing (LSH) techniques [8,16] did not succeed. We discuss the reasons in Section 3.3.

Real-world cryptanalysis of medium-sized instances. For building trust in the bit-security level of cryptographic instances, it is crucial to solve medium-sized instances, e.g. with 60 bit security. This gives us stable data points from which we can more reliably extrapolate to high security levels. An example of good cryptanalysis practice is the break of RSA-768 [13] that allows us to precisely estimate the security of RSA-1024.

Before our work, for McEliece the record code length $n = 1161$ on decodingchallenge.org was reported by Narisada, Fukushima, Kiyomoto with an estimated bit-security level of 56.0. We add two new records McEliece-1223 and McEliece-1284 with estimated bit-security levels of 58.3 and 60.7, respectively. These record computations took us approximately 5 CPU years and 22 CPU years.

As a small technical ingredient to further speed up our new MMT/BJMM implementation, we show how to use the parity of ω to increase the information set size by 1, which saved us approximately 9% of the total running time.

For the quasi-cyclic setting we improved the previously best code length 1938 of Bossard [3] with 6400 CPU days to the five new records 2118, 2306, 2502, 2706, and 2918. The last has a bit-security level of 58.6, and took us (only) 1700 CPU days.

As a technical contribution for the quasi-cyclic setting, we show how to properly generalize the Decoding-One-Out-of-Many (DOOM) strategy to the setting of tree-based enumeration-dominated ISD algorithms. Implementing our DOOM strategy gave us roughly a $\sqrt{n-k}$ experimental speedup, where $n - k = n/2$ is the co-dimension in the quasi-cyclic setting. This coincides with our theoretical analysis, see Section 5.1.

Our real-world cryptanalysis shows that memory access certainly has to be taken into account when computing bit-security, but it might be less costly than suggested. More precisely, our ISD implementations support the so-called *logarithmic cost model*, where an algorithm with time T and memory M has cost $T \cdot \log_2 M$.

[3] https://github.com/FloydZ/decoding

Solid bit-security estimations for McEliece and BIKE/HQC. Based on our record computations and further extensive benchmarking for larger dimensions, we extrapolate to the proposed round-3 McEliece and BIKE/HQC instances. To this end, we also estimate via benchmarking the complexity of breaking AES-128 (NIST Category 1), AES-192 (Category 3) and AES-256 (Category 5) on our hardware.

For McEliece, we find that in the logarithmic cost model the Category 1 instance `mceliece348864` achieves quite precisely the desired 128-bit security level, whereas the Category 3 instance (`mceliece460896`) and two out of three Category 5 instances (`mceliece6688128` and `mceliece6960119`) fail to reach their security level by roughly 20 and 10 bit, even when restricting our attacks to a memory upper limit of $M \leq 2^{80}$. Hence, these instances seem to overestimate security.

For BIKE/HQC, our extrapolation shows that the proposed round-3 instances achieve their desired bit-security levels quite accurately.

Discussion of our results. In our opinion, the appearance of a small security gap for McEliece and no security gap for BIKE/HQC is due to the different weight regimes. Whereas BIKE/HQC use small weight $\omega = \sqrt{n}$, McEliece relies on Goppa codes with relatively large weight $\omega = \Theta(n/\log n)$,

Both the BIKE/HQC and McEliece teams use the asymptotic formula from Equation (2) to analyze their bit-security, which is the more accurate the smaller the weight ω. Hence, while in the BIKE/HQC setting the speedups that we achieve from enumeration-dominated ISD in practice are compensated by other polynomial factors (e.g. Gaussian elimination), in McEliece's (large) weight regime the speedups are so significant that they indeed lead to measurable security losses.

Comparison to previous security estimates. Baldi et al. [4] and more recently Esser and Bellini [11] already provide concrete bit security estimates for code-based NIST candidates. Further, Esser and Bellini introduce new variations of the BJMM and MMT algorithm based on nearest neighbor search, which however did not result in practical gains for our implementation (see Section 3.3 for details).

Both works [4, 11] take into account memory access costs. While [4] uses a logarithmic cost model, [11] considers three models (constant, logarithmic, and cube-root). As opposed to our work, [4, 11] both solely rely on the computation of runtime formulas.

Our work extends and specifies these estimates in the following way. For the first time, we establish with our record computations solid experimental data points for the hardness of instances with roughly 60 bits security. Moreover, our implementation for the first time allows us to identify a proper memory access model that closely matches our experimental observations. Based on our data points, we extrapolate to NIST parameters of cryptographic relevance, using an estimator like [11] with the proper memory access model choice. This eventually allows for a much more reliable security estimate.

2 The MMT/BJMM Algorithm

Let us briefly recap the MMT and BJMM algorithm. From an algorithmic point of view both algorithms are the same. The benefit from BJMM over MMT comes from allowing a more fine-grained parameter selection. In our practical experiments, we mainly used the simpler MMT parameters. Therefore, we refer to our implementation as MMT in the simple parameter setting, and as BJMM in the fine-grained parameter setting.

Main idea. Let $H\mathbf{e} = \mathbf{s}$ be our syndrome decoding instance with parity check matrix $H \in \mathbb{F}_2^{(n-k)\times n}$, unknown error $\mathbf{e} \in \mathbb{F}_2^n$ of known Hamming weight ω, and syndrome $\mathbf{s} \in \mathbb{F}_2^{n-k}$.

As usual in information set decoding, we use some permutation matrix $P \in \mathbb{F}_2^{n\times n}$ to send most of the weight ω to the information set. Let $\bar{H} = HP$ and $\bar{\mathbf{e}} = P^{-1}\mathbf{e}$. Then, obviously $\mathbf{s} = \bar{H}\bar{\mathbf{e}}$.

MMT/BJMM now computes the *semi-systematic form* as originally suggested by Dumer [10]. To this end, fix some parameter $\ell \leq n-k$. Let $\bar{\mathbf{e}} = (\mathbf{e}_1, \mathbf{e}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{k+\ell}$, and assume for ease of exposition that the first $n-k-\ell$ columns of $\bar{H}$ form a full rank matrix. Then we can apply a Gaussian elimination $G \in \mathbb{F}_2^{(n-k)\times(n-k)}$ that yields

$$\bar{\mathbf{s}} := G\mathbf{s} = G\bar{H}\bar{\mathbf{e}} = \begin{pmatrix} I_{n-k-\ell} & H_1 \\ 0 & H_2 \end{pmatrix} = (\mathbf{e}_1 + H_1\mathbf{e}_2, H_2\mathbf{e}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{\ell}. \quad (3)$$

Let $\bar{\mathbf{s}} = (\mathbf{s}_1, \mathbf{s}_2) \in \mathbb{F}_2^{n-k-\ell} \times \mathbb{F}_2^{\ell}$. From Equation (3) we obtain the identity $\mathbf{s}_2 = H_2\mathbf{e}_2$. MMT/BJMM constructs $\mathbf{e}_2$ of weight p satisfying $\mathbf{s}_2 = H_2\mathbf{e}_2$. Notice that for the correct $\mathbf{e}_2$ we directly obtain from Equation (3) that

$$\mathbf{e}_1 = \mathbf{s}_1 + H_1\mathbf{e}_2. \quad (4)$$

Since we know that $\mathrm{wt}(\mathbf{e}_1) = \omega - p$, MMT/BJMM checks for correctness of $\mathbf{e}_2$ via $\mathrm{wt}(\mathbf{s}_1 + H_1\mathbf{e}_2) \stackrel{?}{=} \omega - p$.

Tree-based recursive construction of $\mathbf{e}_2$ using representations. For the tree-based construction of $\mathbf{e}_2$ the reader is advised to closely follow Figure 1. Here, we assume at least some reader's familiarity with the representation technique, otherwise we refer to [12,15] for an introduction.

We write $\mathbf{e}_2$ as a sum $\mathbf{e}_2 = \mathbf{x}_1 + \mathbf{x}_2$ with $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{F}_2^{k+\ell}$ and $\mathrm{wt}(\mathbf{x}_1) = \mathrm{wt}(\mathbf{x}_2) = p_1$. In MMT we choose $p_1 = p/2$, whereas in BJMM we allow for $p_1 \geq p/2$ s.t. a certain amount of one-coordinates in $\mathbf{x}_1, \mathbf{x}_2$ has to cancel in their $\mathbb{F}_2$-sum.

The number of ways to represent the weight-p $\mathbf{e}_2$ as a sum of two weight-p_1 $\mathbf{x}_1, \mathbf{x}_2$, called the number of *representations*, is

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

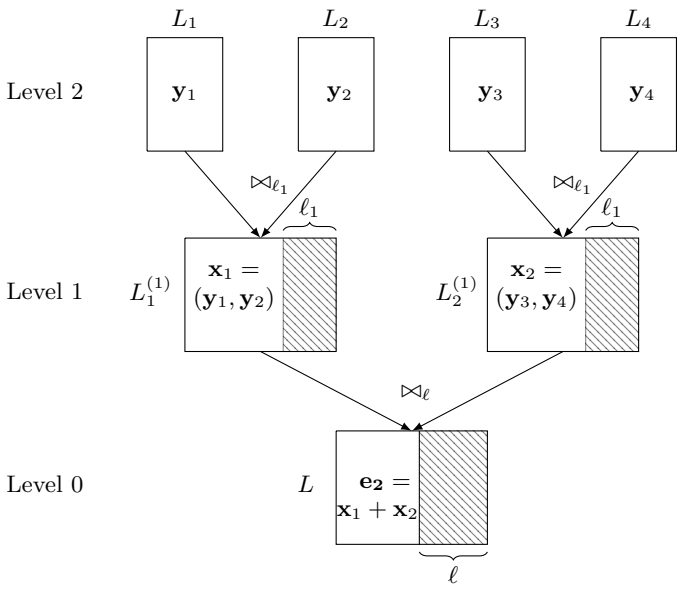

Fig. 1: Search tree of the MMT algorithm. Striped areas indicate matching of the last coordinates of $H\mathbf{x}_i$ or $H(\mathbf{x}_1 + \mathbf{x}_2)$ with some predefined values.

However, it suffices to construct $\mathbf{e}_2$ from a single representation $(\mathbf{x}_1, \mathbf{x}_2)$. Recall from Equation (4) that we have

$$H_2\mathbf{x}_1 = H_2\mathbf{x}_2 + \mathbf{s}_2 \in \mathbb{F}_2^{\ell}.$$

Notice that we do not know the value of $H_2\mathbf{x}_1$. Let us define $\ell_1 := \lfloor \log_2(R) \rfloor$. Since there exist R representations $(\mathbf{x}_1, \mathbf{x}_2)$ of $\mathbf{e}_2$, we expect that for any fixed random target vector $\mathbf{t} \in \mathbb{F}_2^{\ell_1}$ and any projection $\pi : \mathbb{F}_2^{\ell} \to \mathbb{F}_2^{\ell_1}$ on ℓ_1 coordinates (e.g. the last ℓ_1 bits), there is on expectation at least one representation $(\mathbf{x}_1, \mathbf{x}_2)$ that satisfies

$$\pi(H_2\mathbf{x}_1) = \mathbf{t} = \pi(H_2\mathbf{x}_2 + \mathbf{s}_2).$$

We construct all $\mathbf{x}_1$ satisfying $\pi(H_2\mathbf{x}_1) = \mathbf{t}$ in a standard Meet-in-the-Middle fashion. To this end, we enumerate vectors of length $\frac{k+\ell}{2}$ and weight $p_2 := \frac{p_1}{2}$ in baselists L_1, L_2. Analogously, we find all $\mathbf{x}_2$ that satisfy $\pi(H_2\mathbf{x}_2 + \mathbf{s}_2)$ via a Meet-in-the-Middle from baselists L_3, L_4, see Figure 1.

The resulting MMT/BJMM algorithm is described in Algorithm 1.

Runtime analysis. For every permutation P, MMT/BJMM builds the search tree from Figure 1. P has to send weight $\omega - p$ to the information set of size $n - k - \ell$ which happens with probability

$$q := \Pr[P \text{ good}] = \frac{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p}}{\binom{n}{\omega}} \quad . \tag{5}$$

Algorithm 1: MMT Algorithm

Input : $H \in \mathbb{F}_2^{(n-k)\times n}, \mathbf{s} \in \mathbb{F}_2^{n-k}, w \in \mathbb{N}$
Output : $\mathbf{e} \in \mathbb{F}_2^n, H\mathbf{e} = \mathbf{s}$

begin
  Choose optimal ℓ, p, p_2
  Set $\ell_1 = \lfloor \binom{p}{p/2}\binom{k+\ell-p}{p_1-p/2} \rfloor$ and $p_1 = 2p_2$
  repeat
    choose random permutation matrix P
    $\bar{H} = \begin{pmatrix} I_{n-k-\ell} & H_1 \\ 0 & H_2 \end{pmatrix} = GHP$ in semi-systematic form
    $\bar{\mathbf{s}} = (\mathbf{s}_1, \mathbf{s}_2) = G\mathbf{s}$
    Compute
    $$L_1 = L_3 = \{(\mathbf{y}_1, H_2\mathbf{y}_1) \mid \mathbf{y}_1 \in \mathbb{F}_2^{(k+\ell)/2} \times 0^{(k+\ell)/2}, \mathrm{wt}(\mathbf{y}_1) = p_2\}$$
    $$L_2 = \{(\mathbf{y}_2, H_2\mathbf{y}_2) \mid \mathbf{y}_2 \in 0^{(k+\ell)/2} \times \mathbb{F}_2^{(k+\ell)/2}, \mathrm{wt}(\mathbf{y}_2) = p_2\}$$
    $$L_4 = \{(\mathbf{y}_2, H_2\mathbf{y}_2 + \mathbf{s}_2) \mid \mathbf{y}_2 \in 0^{(k+\ell)/2} \times \mathbb{F}_2^{(k+\ell)/2}, \mathrm{wt}(\mathbf{y}_2) = p_2\}$$
    Choose some random $t \in \mathbb{F}_2^{\ell_1}$
    Compute
    $L_1^{(1)} = \{(\mathbf{x}_1, H_2\mathbf{x}_1) \mid \pi(H_2\mathbf{x}_1) = \mathbf{t}, \mathbf{x}_1 = \mathbf{y}_1 + \mathbf{y}_2\}$ from L_1, L_2
    $L_2^{(1)} = \{(\mathbf{x}_2, H_2\mathbf{x}_2 + \mathbf{s}_2) \mid \pi(H_2\mathbf{x}_2 + \mathbf{s}_2) = \mathbf{t}, \mathbf{x}_2 = \mathbf{y}_1 + \mathbf{y}_2\}$ from L_3, L_4
    Compute $L = \{\mathbf{e}_2 \mid H_2\mathbf{e}_2 = \mathbf{s}_2, \mathbf{e}_2 = \mathbf{x}_1 + \mathbf{x}_2\}$ from $L_1^{(1)}, L_2^{(2)}$
    for $\mathbf{e}_2 \in L$ **do**
      $\mathbf{e}_1 = H_1\mathbf{e}_2 + \mathbf{s}_1$
      if $\mathrm{wt}(\mathbf{e}_1) \leq \omega - p$ **then**
        return $P^{-1}(\mathbf{e}_1, \mathbf{e}_2)$
    end
  end
end

The tree construction works in time T_{list}, which is roughly linear in the maximal list size in Figure 1. Let $|L_i|$ denote the common list base size. Then it is not hard to see that the overall expected runtime can be bounded by

$$T = q^{-1} \cdot \tilde{\mathcal{O}}\left(T_{\mathrm{list}}\right), \text{ where } T_{\mathrm{list}} = \max\left\{|L_i|, \frac{|L_i|^2}{2^{\ell_1}}, \frac{|L_i|^4}{2^{\ell+\ell_1}}\right\}.$$

Part of the strength of our MMT/BJMM implementation in the subsequent section is to keep the polynomial factors hidden in the above $\tilde{\mathcal{O}}(\cdot)$-notion small, e.g. by using a suitable hash map data structure.

Locality-Sensitive Hashing (LSH). Most recent improvements to the ISD landscape [8, 16] use nearest neighbor search techniques to speed-up the search-tree computation. We also included LSH techniques in our implementation. However

for the so far benchmarked code dimensions, LSH did not (yet) lead to relevant speedups. See Section 3.3 for further discussion on LSH.

3 Implementing MMT/BJMM Efficiently

In Section 3.1 we introduce an elementary, but at least for McEliece practically effective decoding trick. We then detail our MMT/BJMM implementation in Section 3.2

3.1 Parity Bit Trick

Let us introduce a small technical trick to speed up ISD algorithms, whenever the weight of the error vector is known. Known error weight is the standard case in code-based cryptography. The trick is so elementary that we would be surprised if it was missed in literature so far, but we failed to find a reference, let alone some proper analysis.

Let $H\mathbf{e} = \mathbf{s}$ be our syndrome decoding instance, where ω is the known error weight of $\mathbf{e}$. Then certainly

$$\langle 1^n, \mathbf{e} \rangle = \omega \mod 2.$$

Thus, we can initially append to the parity-check matrix the row vector 1^n, and append to $\mathbf{s}$ the parity bit $\omega \bmod 2$.

Notice that this *parity bit* trick increases the co-dimension by 1, and therefore also the size of the information set. For Prange's permutation-dominated ISD this results in a speedup of

$$\frac{\binom{n}{\omega}}{\binom{n-k}{\omega}} \cdot \frac{\binom{n-k+1}{\omega}}{\binom{n}{\omega}} = \frac{\binom{n-k+1}{\omega}}{\binom{n-k}{\omega}} \ .$$

The speedup for Prange with *parity bit* is the larger the smaller our co-dimension $n-k$ is. For McEliece with small co-dimension and our new record instance $(n = 1284, k = 1028, \omega = 24)$ we obtain more than a 10% speedup, and for the proposed round-3 McEliece parameter sets it is in the range 8-9 %. If instead of Prange's algorithm we use the MMT/BJMM variant that performed best in our benchmarks then the speedup is still in practice a remarkable 9% for the $n = 1284$ instance, and 6-7% for the round-3 parameter sets.

For BIKE and HQC with large co-dimension $n - k = \frac{n}{2}$ and way bigger n, the speedup from the *parity bit* goes down to only 0.5-1%.

3.2 Implementation

Parameter Selection and Benchmarking. As seen in Section 3 and Algorithm 1, the MMT/BJMM algorithm —even when limited to depth 2 search trees— still has to be run with optimized parameters for the weights on all levels of the search tree, and an optimized ℓ. We used an adapted formula based on

the syndrome decoding estimator tool by Esser and Bellini [11] that precisely reflects our implementation to obtain initial predictions for those parameters on concrete instances. We then refined the choice experimentally.

To this end, we measure the number of iterations per second our cluster is able to process for a specific parameter configuration. We then calculate the expected runtime to solve the instance as the number of expected permutations q^{-1} (from Equation (5)) divided by the number of permutations per second. We then (brute-force) searched for an optimal configuration in a small interval around the initial prediction that minimizes the expected runtime.

For instances with McEliece code length $n \leq 1350$ we find optimality of the most simple non-trivial MMT weight configuration with weight $p_2 = 1$ for the baselists $L_1, \ldots, L_4$ on level 2, weight $p_1 = 2$ in level 1, and eventually weight $p = 4$ on level 0 in Figure 1. We refer to the weight configuration $p_2 = 1$ in the baselists as the *low-memory* configuration. Recall that for $p_2 = 0$ MMT becomes Prange's algorithm, and therefore is a memory-less algorithm.

We call configurations with $p_2 \in \{2, 3\}$ *high-memory* configurations. The choice $p_2 = 3$ already requires roughly 40 gigabytes of memory. Increasing the weight to $p_2 = 4$ would increase the memory consumption by another factor of approximately 2^{11}.

Gaussian Elimination. For the Gaussian elimination step we use an open source version [1] of the *Method of the four Russians for Inversion* (M4RI), as already proposed by Bernstein et al. and Peters [7,17]. According to [5] the M4RI algorithm is preferable to other advanced algorithms like Strassen [20] up to matrices of dimension six-thousand. We extended the functionality of [1] to allow for performing a transformation to semi-systematic form, without fully inverting the given matrix. Even for small-memory configurations the permutation and Gaussian elimination step together only account for roughly 2-3% of our total computation time. Therefore we refrain from further optimizations of this step, as introduced in [7,17].

Search Tree Construction. To save memory, we implemented the search tree from Figure 1 in a streaming fashion, as already suggested by Wagner in [25]. See Figure 2 for an illustration showing that we have to store only two baselists and one intermediate list.

Our implementation exploits that $L_1 = L_3$, and L_2 and L_4 only differ by addition of $\mathbf{s}_2$ to the label $H_2\mathbf{y}_2$. To compute the join of L_1, L_2 to $L_1^{(1)}$ we hash list L_1 into a hashmap H_{L_1} using $\pi(H_2\mathbf{y}_1)$ as an index. Then we search each label $\pi(H_2\mathbf{y}_2)$ of list L_2 in H_{L_1}, and store all resulting matches in another hashmap $H_{L_1^{(1)}}$ using the remaining $\ell - \ell_1$ bits of label $H_2\mathbf{x}_2$.

For the right half of the tree we reuse the hashmap H_{L_1} and the list L_2, to which we add $\mathbf{s}_2$. The resulting matches from $L_2^{(1)}$ are directly processed on-the-fly with $H_{L_1^{(1)}}$, producing $\mathbf{e}_2 \in L$.The candidates $\mathbf{e}_2$ are again processed on the fly, and checked whether they lead to the correct counterpart $\mathbf{e}_1$.

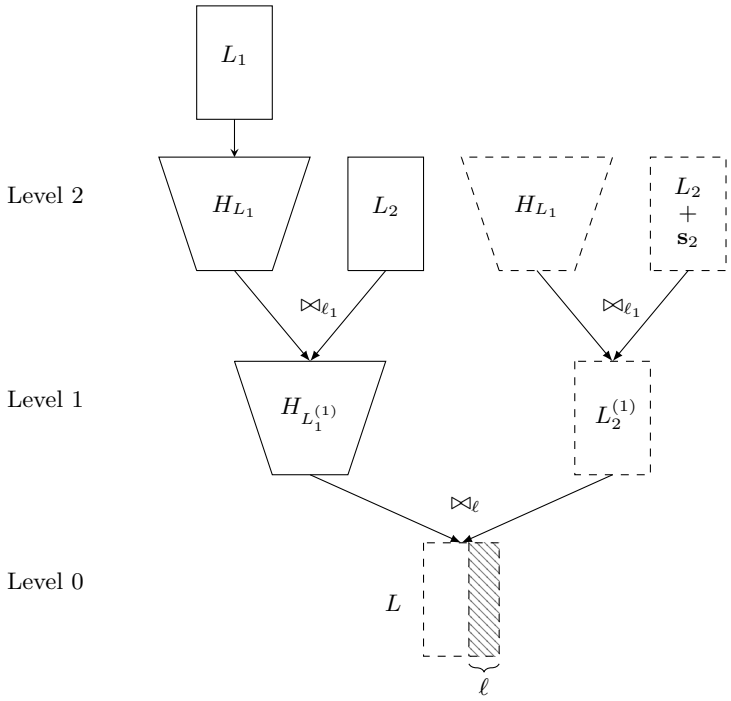

Fig. 2: Streaming implementation of the MMT / BJMM search tree in depth two using two physically stored lists and hashmaps. H_{L_1} and $H_{L_1^{(1)}}$ denote the hashmaps, while dashed lists and hashmaps are processed on the fly.

For speed optimization we worked with a single 64-bit register computation of our candidate solutions throughout all levels of the tree. Even eventually falsifying incorrect $\mathbf{e}_1$ can be performed within 64 bit most of the time. For construction of the baselists L_1, L_2 we used a Gray-code type enumeration.

Parallelization of Low- and High-Memory Configuration. Recall that ISD algorithms consist of a permutation and an enumeration part. In the low-memory regime, we only perform a light enumeration with $p_2 = 1$. The algorithmic complexity is in this configuration dominated by the number of permutations. Therefore, we choose to fully parallelize permutations, i.e., each thread computes its own permutation, Gaussian elimination, and copy of the search tree.

In the high memory regime however, the number of permutations is drastically reduced at the cost of an increasing enumeration complexity. Therefore for the $p_2 = 2, 3$ configurations we choose to parallelize the search tree construction. To this end, we parallelize among N threads by splitting the baselist into N chunks of equal size. To prevent race conditions, every bucket of a hashmap is

also split in N equally sized partitions, where only thread number i can insert into partition i.

3.3 Other Benchmarked Variants – Depth 3 and LSH

It is known that asymptotically, and in the high error regime, an increased search tree depth and the use of LSH techniques [8, 16] both yield asymptotic improvements. We implemented these techniques, but for the following reasons we did not use them for our record computations.

The estimates for depth 2 and 3 complexities are rather close, not giving clear favor to depth 3. This explains why in practice the overhead of another tree level outweighs its benefits.

LSH allows to save on some permutations at the cost of an increased complexity of computing L from the level-one lists $L_1^{(1)}, L_2^{(2)}$. Accordingly, the LSH savings lie in the Gaussian elimination, the base list construction and the matching to level one. Our benchmarks reveal that these procedures together only account for 10-15% of the total running time in the low-memory setting. Moreover, LSH is not well compatible with our streaming design. Therefore, LSH did not yet provide speedups for our computations, but this will likely change for future record computations, see the discussion in Section 4.2.

4 McEliece Cryptanalysis

In this section we give our experimental results on McEliece instances. Besides giving background information on our two record computations, we discuss how good different memory cost models fit our experimental data.

Moreover, we show that MMT reaches its asymptotics *slowly from below*, which in turn implies that purely asymptotic estimates tend to overestimate bit security levels. We eloborate on how to properly estimate McEliece bit security levels in Section 7.

For our computations we used a cluster consisting of two nodes, each one equipped with 2 AMD EPYC 7742 processors and 2 TB of RAM. This amounts for a total of 256 physical cores, allowing for a parallelization via 512 threads.

4.1 Record Computations

Table 1 states the instance parameters of our records we achieved in the McEliece-like decoding category of decodingchallenge.org.

McEliece-1223. We benchmarked an optimal MMT parameter choice of $(\ell, \ell_1, p, p_2) = (17, 2, 4, 1)$. With this low-memory configuration our computing cluster processed $2^{33.32}$ permutations per day, which gives an expected computation time of 8.22 days, since in total we expect $2^{36.36}$ permutations from Equation (5). We solved the instance in 2.45 days, only 30% of the expected

n	k	ω	time (days)	CPU years	bit complexity
1223	979	23	2.45	1.71	58.3
1284	1028	24	31.43	22.04	60.7

Table 1: Parameters of the largest solved McEliece instances, needed wallclock time, CPU years and bit complexity estimate.

running time. If we model the runtime as a geometrically distributed random variable with parameter $q = 2^{-36.36}$, then we succeed within 30% of the expectation with probability 26%.

McEliece-1284. Our benchmarks identified the same optimal parameter set $(\ell, \ell_1, p, p_2) = (17, 2, 4, 1)$ as for McEliece-1223. For this configuration our estimator formula yields an expected amount of $2^{38.49}$ permutations. We benchmarked a total performance of $2^{33.26}$ permutations per day, leading to an expected 37.47 days. We solved the challenge within 31.43 days which is about 84% of the expected running time, and happens with probability about 57%.

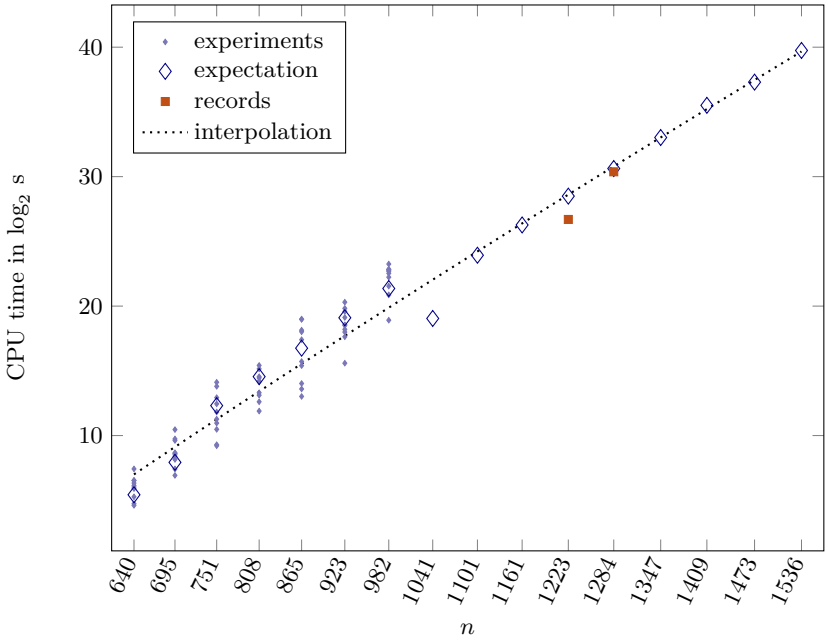

Fig. 3: Running time of experiments and records as well as interpolation for McEliece.

Experimental Results and Discussion. In Figure 3, we plot our record computations as squares. Before we performed our record computations, we heavily tested our implementation with smaller instances $n < 1000$. As before, we computed the expected running time for every value of n, denoted as larger open diamonds in Figure 3, via the quotient of expected permutations and permutations per second on our cluster. The small diamonds depict the actual data points which cluster around their expectation, as desired.

The runtime jumps from $n = 695$ to $n = 751$ and from $n = 982$ to $n = 1041$ can be explained by the instance generation method. For every choice of n the parameters k and ω are derived on decodingchallenge.org as (see [3]) $k = \left\lceil \frac{4n}{5} \right\rceil$ and $\omega = \left\lceil \frac{n}{5\lceil \log n \rceil} \right\rceil$.

For most consecutive instances ω increases by one, but for $n = 695$ to $n = 751$ there is an increase of 2, whereas for $n = 982$ to $n = 1041$ there is a decrease of 1. Besides these jumps, the instance generation closely follows the Classic McEliece strategy.

Comparison with other Implementations. We also compare our implementation to those of Landais [14] and Vasseur [24]. These implementations were used to break the previous McEliece challenges, with the only exception of the $n = 1161$ computation by Narisada, Fukushima, and Kiyomoto that uses non-publicly available code. We find that our implementation performs 12.46 and 17.85 times faster on the McEliece-1284 challenge and 9.56 and 20.36 times faster on the McEliece-1223 instance than [14] and [24], respectively.

4.2 The Cost of Memory

Not very surprising, our experimental results show that large memory consumption leads to practical slowdown. This is in line with the conclusion of the McEliece team [9] that a constant memory access cost model, not accounting for any memory costs, underestimates security. However, this leaves the question how to properly penalize an algorithm with running time T for using memory M. Most prominent models use logarithmic, cube-root or square-root penalty factors, i.e. costs of $T \cdot \log M$, $T \cdot \sqrt[3]{M}$ or $T \cdot \sqrt{M}$, respectively.

In [11] it was shown that logarithmic costs do not heavily influence parameter selection of enumeration-based ISD algorithms, whereas cube-root costs let the MMT advantage deteriorate. Thus, it is crucial to evaluate which cost model most closely matches experimental data.

Break-Even Point for High-Memory Regime. Using our estimator formula we find that under cube-root memory access costs the point where the low-memory configuration $p_2 = 1$ becomes inferior lies around $n = 6000$, falling in the 256-bit security regime of McEliece. In contrast, the logarithmic cost model predicts the break even point at $n \geq 1161$.

By benchmarking the running time of our implementation in the range $n = 1101$ to 1536 for different choices of p_2, see Figure 4, we experimentally

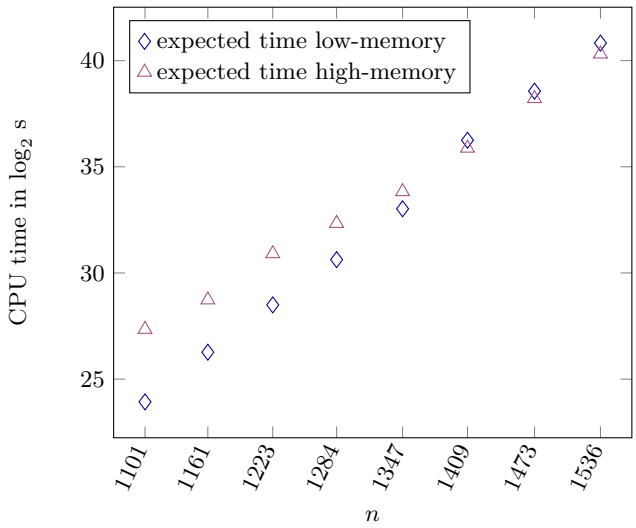

Fig. 4: Estimated running times for low- and high-memory configurations.

find a break even point at $n \approx 1400$. For $n \geq 1400$ the choice $p_2 = 3$ performs best. The configuration $p_2 = 2$ was experimentally always inferior to $p_2 = 1$ and $p_2 = 3$ (which is consistent with our estimation). The reason is that as opposed to $p_2 = 2$ the configuration $p_2 = 3$ does allow for a BJMM parameter selection with $p = 8 < 4p_2$, and also leads to a better balancing of list sizes in the search tree.

In conclusion, the experimentally benchmarked break-even point is way closer to the theoretical point of $n = 1161$ in the logarithmic cost model than to $n = 6000$ in the cube-root model. This already supports the use of logarithmic costs, especially when we take into account that many of our implementation details heavily reward the use of low-memory configurations, such as:

- *Large L3 Caches.* Our processors have an exceptionally large L3 cache of 256 MB that is capable of holding our complete lists in low-memory configurations.
- *Use of Hashmaps.* As indicated in Section 3.2, our parallelization is less effective e.g. for hashmaps in the large-memory regime.
- *Communication complexity.* As opposed to low-memory configurations the high-memory regime requires thread communication for parallelization.

4.3 McEliece Asymptotics: From Above and from Below

It was analyzed in [21], that asymptotically all ISD algorithms converge for McEliece instances to Pranges complexity bound

$$\left(1 - \frac{k}{n}\right)^{\omega}, \text{ see Equation (2).}$$

Since we have rate $\frac{k}{n} = 0.8$ for the decodingchallenge.org parameters, we expect an asymptotic runtime of

$$T(n) = 2^{2.32\frac{n}{\log n}}. \tag{6}$$

This asymptotic estimates supresses polynomial factors. Thus, in Prange's algorithm we have rather $2^{2.32(1+o(1))\frac{n}{\log n}}$, and the algorithm converges to Equation (6) from *above*. For other advanced ISD algorithms the asymptotics suppresses polynomial runtime factors as well as second order improvements. Thus, they have runtime $2^{2.32(1\pm o(1))\frac{n}{\log n}}$, and it is unclear whether they converge from above or below.

Let us take the interpolation line from our data in Figure 3, where we use for the runtime exponent the model function $f(n) = a \cdot \frac{n}{5 \log n} + b$. The interpolation yields

$$a = 2.17 \text{ and } b = -22.97,$$

where the negative b accounts for instances which can be solved in less than a second. The small slope a experimentally demonstrates that the convergence is clearly from *below*, even including realistic memory cost.

However, we still want to find the most realistic memory cost model. To this end, we used our estimator for all instances from Figure 3 in the three different memory access models, constant, logarithmic and cube-root. The resulting bit complexities are illustrated in Figure 5 in a range $n \in [640, 1536]$ for which in practice we have optimal $p_2 \leq 3$. For each model we computed the interpolation according to $f(n) = a \cdot \frac{n}{5 \log n} + b$. For a constant access cost we find $a = 2.04$, for a logarithmic $a = 2.13$, and for the cube-root model we find $a = 2.24$. Hence, again a logarithmic access cost most accurately models our experimental data.

Cryptographic Parameters. So far, we considered only instances with $n \leq 1536$. However, the current round 3 McEliece parameters reach up to code length $n = 8192$. Thus, we also used our estimator to check the slopes a in this cryptographic regime. We compared the ISD algorithms of Prange, Stern and our MMT/BJMM variant. For all algorithms we imposed logarithmic memory access costs $T \cdot \log M$ and considered the three cases of unlimited available memory, as well as 2^{80}-bit and 2^{60}-bit as memory limitation for M. The results for the exponent model $f(n) = a \cdot \frac{n}{5 \log n} + b$ are given in Table 2.

	Prange	Stern	MMT
unlimited	2.438	2.297	2.075
$M \leq 2^{80}$	2.438	2.299	2.207
$M \leq 2^{60}$	2.438	2.308	2.287

Table 2: Slope of interpolation of bitcomplexities under logarithmic memory access costs considering instances with $n \leq 8192$ according to the model function $f(n) = a \cdot \frac{n}{5 \log n} + b$.

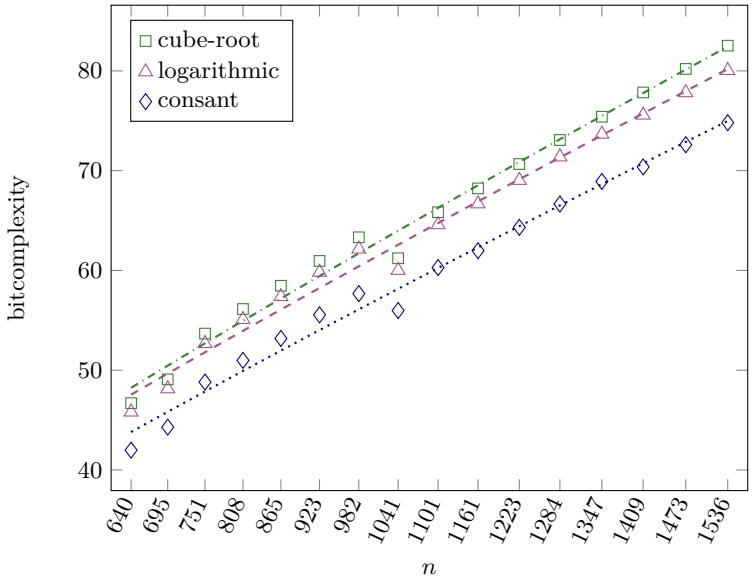

Fig. 5: Estimated bitcomplexities for different memory access cost models and corresponding interpolations.

We observe that Prange does not quickly converge to Equation (2) from above. For Stern and MMT however we are even in the most restrictive memory setting below the exponent from Equation (2). This clearly indicates an overestimate of McEliece security using Equation (2). We eloborate on this more qualitatively in Section 7.

5 The Quasi-Cyclic Setting: BIKE and HQC

The proposals of BIKE and HQC —both alternate finalists of the NIST PQC competition— use double circulant codes with code rate $\frac{1}{2}$, i.e., $n = 2k$. It has been shown by Sendrier [19] that these codes allow for a speedup of Stern's ISD algorithm by a factor of up to $\sqrt{k}$. The basic observation is that the cyclicity immediately introduces k instances of the syndrome decoding problem, where a solution to any of the k instances is a cyclic rotation of the original solution. Thus, this technique is widely known as *Decoding One Out of Many* (DOOM).

Let $H_1, H_2 \in \mathbb{F}_2^{k \times k}$ be two circulant matrices satisfying

$$\left(H_1 \; H_2\right)(\mathbf{e}_1, \mathbf{e}_2) = \mathbf{s}$$

with $\mathbf{e}_1, \mathbf{e}_2 \in \mathbb{F}_2^k$. Let us denote by $\mathrm{rot}_i(\mathbf{x})$ the cyclic left rotation of $\mathbf{x}$ by i positions. Then for any $i = 0, \ldots, k-1$ we have

$$\left(H_1 \; H_2\right)(\mathrm{rot}_i(\mathbf{e}_1), \mathrm{rot}_i(\mathbf{e}_2)) = \mathrm{rot}_i(\mathbf{s}) =: \mathbf{s}_i.$$

This implies that a solution to any of the k instances $(H_1H_2, \mathbf{s}_i, \omega)$ yields $(\mathbf{e}_1, \mathbf{e}_2)$.

Note that in the special case of $\mathbf{s} = \mathbf{0}$, thus, when actually searching for a small codeword the instances are all the same, meaning there simply exist k different solutions $\mathbf{e}$. In this case any ISD algorithm obtains a speedup of k.

For $\mathbf{s} \neq \mathbf{0}$ one usually assumes a speedup of $\sqrt{k}$ in the quasi-cyclic setting, referring to Sendrier's DOOM result [19]. However, [19] only analyzes Stern's algorithm.

In the following section we adapt the idea of Sendrier's DOOM to the MMT/BJMM algorithm in the specific setting of double circulant codes, achieving speedups slightly larger than $\sqrt{k}$ both in theory and practical experiments.

5.1 Decoding one out of k ($\mathbf{DOOM}_k$)

To obtain a speedup from the k instances we modify the search tree of our MMT/BJMM variant such that in every iteration all k syndromes are considered. To this end, similar to Sendrier, we first enlarge list L_4 (compare to Figure 2) by exchanging every element $(\mathbf{x}, H\mathbf{x}) \in L_4$ by $(\mathbf{x}, H\mathbf{x} + \bar{\mathbf{s}}_i)$ for all $i = 1 \dots k$, where $\bar{\mathbf{s}}_i := G\mathbf{s}_i$ denotes the i-th syndrome after the Gaussian elimination. This results in a list that is k times larger than L_4. To compensate for this increased list size we enumerate in L_4 initially only vectors of weight $p_2 - 1$ rather than p_2.

This simple change already allows for a speedup of our MMT/BJMM algorithm of order $\sqrt{k}$, as shown in the following lemma.

Lemma 1 ($\mathbf{DOOM}_k$ speedup). *A syndrome decoding instance with double circulant parity-check matrix, code rate $\frac{k}{n} = \frac{1}{2}$ and error weight $\omega = \Theta(\sqrt{k})$ allows for a speedup of the MMT/BJMM algorithm by a factor of $\Omega(\sqrt{k})$.*

Proof. First note, that since list L_4 is duplicated for every syndrome $\mathbf{s}_i$ by the correctness of the original MMT algorithm our modification is able to retrieve any of the rotated solutions if permutation distributed the weight properly.

Let us first analyze the impact of our change on the size of the list L_4. The decrease of the weight of the vectors in L_4 from p_2 to $p_2 - 1$ decreases the size by a factor of

$$\delta_{\mathrm{L}} := \frac{\binom{(k+\ell)/2}{p_2}}{\binom{(k+\ell)/2}{p_2-1}} = \frac{\frac{k+\ell}{2} - p_2 + 1}{p_2} \approx \frac{k}{2 \cdot p_2},$$

since $\ell, p_2 \ll k$. Thus, together with the initial blowup by k for every syndrome we end up with a list that is roughly $2p_2$ times as large as the original list. Next let us study the effect on the probability of a random permutation distributing the error weight properly for anyone of the k error vector rotations, which is

$$\begin{aligned}\delta_P &:= \frac{\binom{n-k-\ell}{\omega-p+1}\binom{k+\ell}{p-1} \cdot k / \binom{n}{\omega}}{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p} / \binom{n}{\omega}} = \frac{\binom{n-k-\ell}{\omega-p+1}\binom{k+\ell}{p-1} \cdot k}{\binom{n-k-\ell}{\omega-p}\binom{k+\ell}{p}} \\ &= \frac{(k-\ell-\omega+p) \cdot p \cdot k}{(\omega-p+1)(k+\ell-p+1)} = \Omega(\sqrt{k}).\end{aligned}$$

	Instance		$\log\left(\sqrt{k}\right)$	Speedup	
	k	ω		Stern	MMT
Challenge-1	451	30	4.41	4.88	4.96
Challenge-2	883	42	4.89	5.39	5.43
QC-2918	1459	54	5.26	5.77	5.77
BIKE-1	12323	134	6.79	7.58	7.47
BIKE-3	24659	199	7.29	8.00	7.55
BIKE-5	40973	264	7.66	8.32	8.06
HQC-1	17669	132	7.05	8.14	8.00
HQC-3	35851	200	7.56	8.55	8.39
HQC-5	57637	262	7.91	8.83	8.66

Table 3: Estimated DOOM_k speedups for Stern and MMT in the quasi-cyclic setting with double circulant codes ($n = 2k$).

Here the denominator states the probability of a permutation inducing the correct weight distribution on any of the k syndromes, while the numerator is the probability of success in any iteration of the MMT algorithm (compare to Equation (5)). Observe that the last equality follows from the fact, that $\omega = \Theta(\sqrt{k})$ and $p \ll \omega$ as well as $\ell \ll k$.

So far we showed, that our modification increases the list size of L_4 by a small factor of $2p_2$, while we enhance the probability of a good permutation for any of the given k instances by a factor of $\Omega(\sqrt{k})$. While in the case of Sterns' algorithm this is already enough to conclude that the overall speedup in this setting is $\Omega(\sqrt{k})$, as long as $p_2 \ll k$, for MMT/BJMM we also need to consider the reduced amount of representations. Note that the amount of representations decreases from an initial R to R_k, i.e., by a factor of

$$\begin{aligned}\delta_R := \frac{R_k}{R} &= \frac{\binom{p-1}{p/2}\binom{k+\ell-p+1}{p_1-p/2}}{\binom{p}{p/2}\binom{k+\ell-p}{p_1-p/2}} \\ &= \frac{(k+\ell-p+1)\cdot p/2}{(k+\ell-p/2-p_1+1)\cdot p} = \frac{(k+\ell-p+1)}{2(k+\ell-p+1-\varepsilon)} \approx \frac{1}{2},\end{aligned}$$

Here $\varepsilon = p_1 - p/2$ is the amount of 1-entries added by BJMM to cancel out during addition, which is usually a small constant. Hence $\ell_1 := \log R$ in Algorithm 1 decreases by one. This in turn increases the time for computing the search-tree by a factor of at most two.

In summary, we obtain a speedup of $\delta_{\mathrm{P}} = \Omega(\sqrt{k})$ on the probability while losing a factor of at most $\frac{\delta_{\mathrm{L}}}{\delta_R} = 4p_2$ in the construction of the tree. Hence, for MMT/BJMM with $p_2 \ll \sqrt{k}$ this yields an overall speedup of $\Omega(\sqrt{k})$. □

We included the DOOM_k improvement in our estimator formulas for Stern as well as MMT. Table 3 shows the derived estimated speedups. As a result both

algorithms Stern and MMT achieve comparable DOOM_k speedups slightly larger than $\sqrt{k}$. Additionally, we performed practical experiments on the instances listed as *Challenge-1* and *Challenge-2* to verify the estimates. Therefore, we solved these instances with MMT with and without the DOOM_k technique. Averaged over ten executions we find speedups of 4.99 and 5.40 respectively (closely matching 4.96 and 5.43 from Table 3).

6 Quasi-Cyclic Cryptanalysis

In the quasi-cyclic setting we obtained five new decoding records on decodingchallenge.org with our MMT implementation [3], see Table 4. Instances are defined on [3] for every ω using parameters $n = \omega^2 + 2$ and $k = \frac{n}{2}$, closely following the BIKE and HQC design.

n	k	ω	time (days)	CPU years	bit complexity
2118	1059	46	0.08	0.05	50.5
2306	1153	48	0.22	0.15	52.5
2502	1459	50	0.30	0.21	54.6
2706	1353	52	1.18	0.83	56.6
2918	1459	54	3.33	2.33	58.6

Table 4: Parameters of the largest solved BIKE/HQC instances, needed wallclock time, CPU years and bit complexity estimates.

QC-2918. The largest instance we were able to solve has parameters $(n, k, w) = (2918, 1459, 54)$, and took us 3.33 days on our cluster. The optimal identified parameter set is $(\ell, \ell_1, p, p_2) = (21, 1, 3, 1)$, for which we estimated $2^{31.9}$ permutations. We were able to perform $2^{29.31}$ permutations per day, resulting in an expected running time of 6.02 days. Our computation took only 55% of the expected time, which happens with probability 42%.

Interpolation. In Figure 6 we give the expected running times that we obtained via benchmarking, both with (diamonds) and without (triangles) our DOOM_k result from Section 5.1. Our five record computations are depicted as squares. All record computations where achieved in a runtime closely matching the expected values.

In the quasi-cyclic setting with rate $\frac{k}{n} = \frac{1}{2}$ and $\omega = \sqrt{n}$ Prange's runtime formula from Equation (2) gives $2^{\sqrt{n}}$. An interpolation of our experimental data points using the model $f(n) = a\sqrt{n} + b$ yields a best fit for

$$f(n) = 1.01\sqrt{n} - 26.42. \tag{7}$$

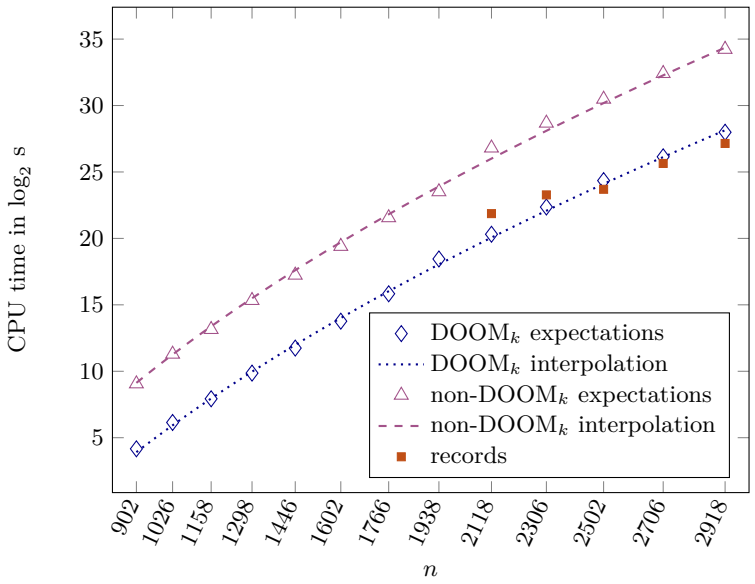

Fig. 6: Estimated running times and interpolations for low- and high-memory configurations.

The slope $a = 1.01$ shows how accurately our MMT implementation matches the asymptotics already for medium sized instances, i.e., our MMT advantage and the polynomial runtime factors almost cancel out.

Concrete vs Asymptotic. Similar to the McEliece setting in Section 4.3 and in Table 2, we also performed for BIKE/HQC an interpolation of estimated bit complexities in the logarithmic cost model using the algorithms of Prange, Stern and our MMT variant. We included instances up to code length $120,000$, reflecting the largest choice made by an HQC parameter set. As opposed to Section 4.3 we do not need addtional memory limitations, since none of the optimal configurations exceeds 2^{60}-bit of memory.

The interpolation with $f(n) = a\sqrt{n} + b$ gave us slopes of 1.054, 1.019 and 1.017 for Prange, Stern and MMT, respectively, i.e., all slopes are slightly above the asymptotic prediction of $a = 1$. Thus, as opposed to the McEliece setting our MMT benefits are canceled by polynomial factors.

Verification of the DOOM$_k$ Speedup. From Figure 6, we can also experimentally determine the speedup of our DOOM$_k$ technique inside MMT. Lemma 1 predicts a speedup of $\sqrt{k} = \sqrt{n/2}$. Let $f(n) = 1.01\sqrt{n} - 26.42$ as before, and in addition take the model $f(n) + c \cdot \frac{\log(n/2)}{2}$ for non-DOOM$_k$. The new model should fit with $c = 1$.

The interpolation of our experimental non-DOOM$_k$ data, see the dashed line in Figure 6, yields $c = 1.17$. Thus, in practice we obtain a DOOM$_k$ speedup of $k^{0.58}$, slightly larger than $\sqrt{k}$.

7 Estimating Bit-Security for McEliece and BIKE/HQC

Based on our record computations, let us extrapolate to the hardness of breaking round-3 McEliece, BIKE and HQC. To provide precise statements about the security levels of proposed parameter sets, we also need to compare with the hardness of breaking AES. Recall that NIST provides five security level categories, where the most frequently used categories 1, 3, and 5 relate to AES. Category 1, 3, and 5 require that the scheme is as hard to break as AES-128, AES-192, and AES-256, respectively.

For AES we benchmarked the amount of encryptions per second on our cluster using the openssl benchmark software. The results for different key-lengths are listed in Table 5. For AES-192 and AES-256, we increased the blocklength from 128 to 256 bit, such that on expectation only a single key matches a known plaintext-ciphertext pair.

	AES-128	AES-192	AES-256
10^9enc/sec	2.16	0.96	0.83

Table 5: Number of AES encryptions per second performed by our cluster.

From Table 5 we extrapolate the running time to break AES-128, AES-192, and AES-256 on our hardware.

Extrapolation for McEliece and BIKE/HQC. Let us detail our extrapolation methodology. We take as starting points the real runtimes of $22.04 = 2^{4.46}$ CPU years for McEliece-1284 and 2.33 CPU years for QC-2918.

Then we estimate by which factor it is harder to break the round-3 instances, and eventually compare the resulting runtime to the hardness of breaking AES.

Let us give a numerical example for McEliece-4608. Assume that we take 2^{60}-bit memory limitation for M, and we are in the most realistic logarithmic memory cost model. In this setting our estimator (without LSH) gives for $n = 4608$ a bit complexity of 187.72, and for $n = 1284$ a bit complexity of 65.27. Thus, it is a factor of $2^{122.45}$ harder to break McEliece-4608 than to break our record McEliece-1284. Therefore, we conclude that a break of McEliece-4608 on our hardware would require $2^{4.46} \cdot 2^{122.45} = 2^{126.91}$ CPU years.

In contrast, from Table 5 we conclude that breaking AES-192 on our hardware requires $2^{145.24}$ CPU years. Thus, from our extrapolation McEliece-4608 is a factor of $2^{18.33}$ *easier to break* than AES-192. This is denoted by -18.33 in Table 6.

McEliece Slightly Overestimates Security. For completeness, we consider in Table 6 all three different memory-access cost models, constant, logarithmic and cube-root, even though we identified the *logarithmic* model as most realistic (compare to Section 4.2). Recall that in these models an algorithm with memory M suffers either no penalty (constant), a multiplicative factor of $\log M$ (logarithmic) or even a $\sqrt[3]{M}$ factor penalty (cube-root).

Moreover, we also provide memory limitations for the constant and logarithmic models. This is unnecessary in the cube-root model, in which no optimal parameter configuration exceeds a memory bit complexity of 60.

McEliece		Category 1 $n = 3488$	Category 3 $n = 4608$	Category 5a $n = 6688$	Category 5b $n = 6960$	Category 5c $n = 8192$
constant	unlimited	0.09	−24.86	−23.18	−23.80	6.10
	$M \leq 2^{80}$	1.54	−21.52	−11.67	−10.87	23.37
	$M \leq 2^{60}$	4.80	−19.12	− 3.86	− 3.80	32.70
logarithmic	unlimited	1.77	−23.11	−20.70	−21.29	8.84
	$M \leq 2^{80}$	2.86	−20.41	−10.46	− 9.63	24.64
	$M \leq 2^{60}$	5.55	−18.33	− 3.46	− 3.40	33.16
cube-root		10.37	−12.27	0.82	1.38	38.22

Table 6: Difference in bit complexity of breaking McEliece and corresponding AES instantiation under different memory access cost.

Let T_{McEliece} denote the extrapolated McEliece runtime, and let T_{AES} be the extrapolated AES runtime in the respective security category. Then Table 6 provides the entries $\log_2(\frac{T_{\text{McEliece}}}{T_{\text{AES}}})$. Thus, a negative x-entry indicates that this McEliece instance is x bits easier to break than its desired security category.

Whereas the Category 1 instance McEliece-3488 meets its security level in all memory models, the Category 3 instance McEliece-4608 misses the desired level by roughly 20 bits for constant/logarithmic costs. Even for cube-root costs McEliece-4608 is still 12 bits below the required level.

The Category 5a and 5b McEliece instances are in the realistic logarithmic model with 2^{80}-bit memory also 10 bits below their desired security level, whereas the Category 5c McEliece instance is independent of the memory model above its security level.

BIKE/HQC Accurately Matches Security. In Table 7 we state our results for BIKE and HQC. As opposed to the McEliece setting we do not need memory limitations here, since none of the estimates exceeded 2^{60}-bit of memory.

Note that for BIKE we need to distinguish an attack on the key and an attack on a message. That is because recovering the secret key from the public key

BIKE / HQC			Category 1	Category 3	Category 5
constant	BIKE	message	2.44	2.50	3.49
		key	3.88	2.13	5.87
	HQC		1.24	4.28	2.23
logarithmic	BIKE	message	2.86	3.04	4.10
		key	4.42	3.11	6.74
	HQC		1.72	4.87	2.90
cube-root	BIKE	message	4.47	5.20	6.68
		key	5.77	5.00	9.03
	HQC		3.62	7.34	5.75

Table 7: Difference in bit complexity of breaking BIKE/HQC and corresponding AES instantiation under different memory access cost.

corresponds to finding a low-weight codeword, whereas recovering the message from a ciphertext corresponds to a syndrome decoding instance, where the syndrome is usually not the zero vector. Both settings allow for different speedups as outlined in Section 5.

We observe that the BIKE as well as the HQC instances precisely match their claimed security levels already in the conservative setting of constant memory access costs. Introducing memory penalties only leads to slight increases in the security margins.

References

1. Albrecht, M., Bard, G.: The M4RI Library. The M4RI Team (2021), http://m4ri.sagemath.org
2. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology **9**(3), 169–203 (2015)
3. Aragon, N., Lavauzelle, J., Lequesne, M.: decodingchallenge.org (2019), http://decodingchallenge.org
4. Baldi, M., Barenghi, A., Chiaraluce, F., Pelosi, G., Santini, P.: A finite regime analysis of information set decoding algorithms. Algorithms **12**(10), 209 (2019)
5. Bard, G.V.: Algorithms for solving linear and polynomial systems of equations over finite fields, with applications to cryptanalysis. University of Maryland, College Park (2007)
6. Becker, A., Joux, A., May, A., Meurer, A.: Decoding random binary linear codes in $2^{n/20}$: How $1 + 1 = 0$ improves information set decoding. In: 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
7. 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)

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. 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)
10. Dumer, I.: On minimum distance decoding of linear codes. In: Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory. pp. 50–52 (1991)
11. Esser, A., Bellini, E.: Syndrome decoding estimator. IACR Cryptol. ePrint Arch. **2021**, 1243 (2021)
12. Howgrave-Graham, N., Joux, A.: New generic algorithms for hard knapsacks. In: Gilbert, H. (ed.) Advances in Cryptology – EUROCRYPT 2010. Lecture Notes in Computer Science, vol. 6110, pp. 235–256. Springer, Heidelberg, Germany, French Riviera (May 30 – Jun 3, 2010). https://doi.org/10.1007/978-3-642-13190-5_12
13. Kleinjung, T., Aoki, K., Franke, J., Lenstra, A.K., Thomé, E., Bos, J.W., Gaudry, P., Kruppa, A., Montgomery, P.L., Osvik, D.A., et al.: Factorization of a 768-bit rsa modulus. In: Annual Cryptology Conference. pp. 333–350. Springer (2010)
14. Landais, G.: Code of Grégory Landais (2012), https://gforge.inria.fr/projects/collision-dec/
15. 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
16. 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
17. Peters, C.: Information-set decoding for linear codes over f q. In: International Workshop on Post-Quantum Cryptography. pp. 81–94. Springer (2010)
18. Prange, E.: The use of information sets in decoding cyclic codes. IRE Transactions on Information Theory **8**(5), 5–9 (1962)
19. Sendrier, N.: Decoding one out of many. In: International Workshop on Post-Quantum Cryptography. pp. 51–67. Springer (2011)
20. Strassen, V.: Gaussian elimination is not optimal. Numerische mathematik **13**(4), 354–356 (1969)
21. 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)
22. 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
23. 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
24. Vasseur, V.: Code of Valentin Vasseur (2020), https://gitlab.inria.fr/vvasseur/isd
25. 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