Decoding Random Binary Linear Codes in $2^{n/20}$: How $1 + 1 = 0$ Improves Information Set Decoding

Anja Becker[1], Antoine Joux[1,2], Alexander May[3⋆], and Alexander Meurer[3⋆⋆]

[1] Université de Versailles Saint-Quentin, Laboratoire PRISM
[2] DGA
[3] Ruhr-University Bochum, Horst Görtz Institute for IT-Security
anja.becker@prism.uvsq.fr,antoine.joux@m4x.org
{alex.may,alexander.meurer}@rub.de

Abstract. Decoding random linear codes is a well studied problem with many applications in complexity theory and cryptography. The security of almost all coding and LPN/LWE-based schemes relies on the assumption that it is hard to decode random linear codes. Recently, there has been progress in improving the running time of the best decoding algorithms for binary random codes. The ball collision technique of Bernstein, Lange and Peters lowered the complexity of Stern's information set decoding algorithm to $2^{0.0556n}$. Using *representations* this bound was improved to $2^{0.0537n}$ by May, Meurer and Thomae. We show how to further increase the number of representations and propose a new information set decoding algorithm with running time $2^{0.0494n}$.

1 Introduction

The NP-hard problem of decoding a random linear code is one of the most promising problems for the design of cryptosystems that are secure even in the presence of quantum computers. Almost all code-based cryptosystems, e.g. McEliece, rely on the fact that random linear codes are hard to decode. In order to embed a trapdoor in coding-based cryptography one usually starts with a well-structured secret code C and linearly transforms it into a code C' that is supposed to be indistinguishable from a random code.

An attacker has two options. Either he tries to distinguish the scrambled version C' of C from a random code by revealing the underlying structure, see [10, 27]. Or he directly tries to run a generic decoding algorithm on the scrambled code C'.

Also closely related to random linear codes is the learning parity with noise (LPN) problem that is frequently used in cryptography [1, 13, 16]. In LPN, one directly starts with a random linear code C and the LPN search problem is a decoding problem in C. It was shown in [26] that the popular LPN decision

⋆ Supported by DFG project MA 2536/7-1 and by ICT-2007-216676 ECRYPT II,
⋆⋆ Ruhr-University Research School, Germany Excellence Initiative [DFG GSC 98/1]

variant, a very useful tool for many cryptographic constructions, is equivalent to the LPN search problem, and thus equivalent to decoding a random linear code. The LWE problem of Regev [26] is a generalization of LPN to codes over a larger field. Our decoding algorithm could be adjusted to work for these larger fields (similar to what was done in [8, 25]). Since the decoding problem lies at the the heart of coding-based and LPN/LWE-based cryptography it is necessary to study its complexity in order to define proper security parameters for cryptographic constructions.

Let us start by providing some useful notation. A binary linear code C is a k-dimensional subspace of $\mathbb{F}_2^n$ where n is called the length of the code and $R := \frac{k}{n}$ is called its rate. A random k-dimensional linear code C of length n can be defined as the kernel of a random full-rank matrix $\mathbf{H} \in_R \mathbb{F}_2^{(n-k)\times n}$, i.e. $C = \{\mathbf{c} \in \mathbb{F}_2^n \mid \mathbf{H}\mathbf{c}^t = \mathbf{0}^t\}$. The matrix $\mathbf{H}$ is called a parity check matrix of C. For ease of presentation, we use the convention that all vectors are column vectors which allows as to omit all transpositions of vectors.

The distance d of a linear code is defined by the minimal Hamming distance between two codewords. Hence every vector $\mathbf{x}$ whose distance to the closest codeword $\mathbf{c} \in C$ is at most the error correction capacity $\omega = \lfloor \frac{d-1}{2} \rfloor$ can be uniquely decoded to $\mathbf{c}$.

For any point $\mathbf{x} = \mathbf{c}+\mathbf{e} \in \mathbb{F}_2^n$ that differs from a codeword $\mathbf{c} \in C$ by an error vector $\mathbf{e}$, we define its *syndrome* as $s(\mathbf{x}) := \mathbf{H}\mathbf{x} = \mathbf{H}(\mathbf{c}+\mathbf{e}) = \mathbf{H}\mathbf{e}$. Hence, the syndrome only depends on the error vector $\mathbf{e}$ and not on the codeword $\mathbf{c}$. The *syndrome decoding problem* is to recover $\mathbf{e}$ from $s(\mathbf{x})$. This is equivalent to decoding in C, since the knowledge of $\mathbf{e}$ suffices to recover $\mathbf{c}$ from $\mathbf{x}$.

Usually in cryptographic settings the Hamming weight of $\mathbf{e}$ is smaller than the error correction capability, i.e. $\mathrm{wt}(\mathbf{e}) \leq \omega = \lfloor \frac{d-1}{2} \rfloor$, which ensures unique decoding. This setting is also known as *half/bounded distance decoding*. All known half distance decoding algorithms achieve their worst case behavior for the choice $\mathrm{wt}(\mathbf{e}) = \omega$. As a consequence we assume $\mathrm{wt}(\mathbf{e}) = \omega$ throughout this work. In complexity theory, one also studies the so-called *full decoding* where one has to compute a closest codeword to a given *arbitrary* vector $\mathbf{x} \in \mathbb{F}_2^n$. We also give the complexity of our algorithm for full decoding, but in the following we will focus on half-distance decoding.

The running time of decoding algorithms for linear codes is a function of the three code parameters $[n, k, d]$. However, with overwhelming probability random binary linear codes attain a rate $R := \frac{k}{n}$ which is close to the Gilbert Varshamov bound $1 - H(\frac{d}{n})$ [9]. Therefore, we can express the running time $T(n, R)$ as a function in n, R only. One usually measures the complexity of decoding algorithms asymptotically in the code length n. Since all generic decoding algorithms run in exponential time, a reasonable metric is the complexity

coefficient $F(R)$ as defined in [8], i.e. $F(R) = \lim_{n\to\infty} \frac{1}{n} \log T(n, R)$ which suppresses polynomial factors since $\lim \frac{1}{n} \log p(n) = 0$ for any polynomial $p(n)$. Thus, we have $T(n, R) = 2^{nF(R)+o(n)} \leq 2^{n\lceil F(R)\rceil_\rho}$ for large enough n. We obtain the worst-case complexity by taking $\max_{0<R<1} \lceil F(R)\rceil_\rho$. Here, $\lceil x\rceil_\rho := \lceil x \cdot 10^\rho\rceil \cdot 10^{-\rho}$ denotes rounding up $x \in \mathbb{R}$ to a certain number of $\rho \in \mathbb{N}$ decimal places.

Related work. In syndrome decoding one has to compute $\mathbf{e}$ from $s(\mathbf{x})$, which means that one has to find a weight-ω linear combination of the columns of $\mathbf{H}$ that sums to the syndrome $s(\mathbf{x})$ over $\mathbb{F}_2^{n-k}$. Thus, a brute-force algorithm would require to compute $\binom{n}{\omega}$ column sums. Inspired by the work of Prange [24], it was already mentioned in the original work of McEliece [21] and later more carefully studied by Lee and Brickell [18] that the following approach, called *information set decoding*, yields better complexity.

Information set decoding basically proceeds in two steps, an initial transformation step and a search step. Both steps are iterated in a loop until the algorithm succeeds. The initial transformation step starts by randomly permuting the columns of $\mathbf{H}$. In particular, this permutes the ω columns of $\mathbf{H}$ that sum to $s(\mathbf{x})$, and thus permutes the coordinates of $\mathbf{e}$. Then we apply Gaussian elimination on the rows of $\mathbf{H}$ in order to obtain a systematic form $(\mathbf{Q} \mid \mathbf{I}_{n-k})$, where $\mathbf{Q} \in \mathbb{F}_2^{(n-k)\times k}$ and $\mathbf{I}_{n-k}$ is the $(n-k)$-dimensional identity matrix. The Gaussian elimination operations are also applied to $s(\mathbf{x})$ which results in $\tilde{s}(\mathbf{x})$.

Let us fix an integer $p < \omega$. In the search step, we compute for every linear combination of p columns from $\mathbf{Q}$ its Hamming distance to $\tilde{s}(\mathbf{x})$. If the distance is exactly $\omega - p$ then can we add to our p columns those $\omega - p$ unit vectors from $\mathbf{I}_{n-k}$ that exactly yield $\tilde{s}(\mathbf{x})$. Undoing the Gauss elimination recovers the desired error vector $\mathbf{e}$. Obviously, information set decoding can only succeed if the initial column permutation results in a permuted $\mathbf{e}$ that has exactly p ones in its first k coordinates and $\omega - p$ ones in its last $n - k$ coordinates. Optimization of p leads to a running time of $2^{0.05752n}$.

Leon[19] and Stern[29] observed in 1989 that one can improve on the running time when replacing in the search step the brute-force search for weight-p linear combinations by a Meet-in-the-middle approach. Let us fix an integer $\ell < n - k$ and let us project $(\mathbf{Q} \mid \mathbf{I}_{n-k})$ to its first ℓ rows. We split the projection of $\mathbf{Q}$ into two matrices $\mathbf{Q}_1, \mathbf{Q}_2$ each having $\frac{k}{2}$ columns. Then we create two lists $\mathcal{L}_1, \mathcal{L}_2$ that contain all weight-$\frac{p}{2}$ linear combinations of columns from $\mathbf{Q}_1$ and $\mathbf{Q}_2$, respectively. Moreover, we add the projection of $\tilde{s}(\mathbf{x})$ to every element in $\mathcal{L}_2$ and sort the resulting list.

Then we search for matching elements from $\mathcal{L}_1$ and $\mathcal{L}_2$. These elements define weight-p sums of vectors from $\mathbf{Q}$ that exactly match $\tilde{s}(\mathbf{x})$ in its first ℓ co-

ordinates. As before, if the remaining coordinates differ from $\tilde{s}(\mathbf{x})$ by a weight-$(\omega - p)$ vector, then we can correct these positions by suitable unit vectors from $\mathbf{I}_{n-k}$. The running time of this algorithm is $2^{0.05564n}$.

The ball collision technique of Bernstein, Lange and Peters [4] lowers this complexity to $2^{0.05559n}$ by allowing a non-exact matching of the elements of $\mathcal{L}_1$ and $\mathcal{L}_2$. The same asymptotic complexity can be achieved by transforming $\mathbf{H}$ into $(\mathbf{Q} \mid \begin{smallmatrix}\mathbf{0}\\ \mathbf{I}_{n-k-\ell}\end{smallmatrix})$ with $\mathbf{Q} \in \mathbb{F}_2^{(n-k)\times(k+\ell)}$, as proposed by Finiasz and Sendrier [11]. The lists $\mathcal{L}_1, \mathcal{L}_2$ then each contain all weight-$\frac{p}{2}$ sums out of $\frac{k+\ell}{2}$ columns. The asymptotic analysis of this variant can be found in [22].

Notice that finding a weight-p sum of columns of $\mathbf{Q}$ that exactly matches $\tilde{s}(\mathbf{x})$ in ℓ coordinates is a vectorial version of the subset sum problem in $\mathbb{F}_2^{\ell}$. This vectorial version was called the *column match problem* by May, Meurer and Thomae (MMT) [22], who adapted the subset sum representation technique from Howgrave-Graham and Joux [14] to the column match problem.

Let $\mathbf{Q} \in \mathbb{F}_2^{(n-k)\times(k+\ell)}$ be as before, where $\mathbf{q}_1, \ldots, \mathbf{q}_{k+\ell}$ denote the columns of $\mathbf{Q}$. A Meet-in-the-Middle approach matches the first ℓ coordinates via the identity

$$\sum_{i \in I_1} \mathbf{q}_i = \sum_{i \in I_2} \mathbf{q}_i + \tilde{s}(\mathbf{x}) \ , \tag{1}$$

where $I_1 \subset \left[1, \frac{k+\ell}{2}\right]$, $I_2 \subset \left[\frac{k+\ell}{2} + 1, k + \ell\right]$ and $|I_1| = |I_2| = \frac{p}{2}$.

Using the representation technique, one chooses I_1 and I_2 no longer from half-sized intervals but they both are chosen from the whole interval $[1, k + \ell]$ such that $I_1 \cap I_2 = \emptyset$. Thus, every solution I admits $\binom{p}{p/2}$ representations $I = I_1 \cup I_2$. Notice that increasing the range of I_1, I_2 also increases the size of the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ from $\binom{(k+\ell)/2}{p/2}$ to $\binom{k+\ell}{p/2}$. But constructing only a $\binom{p}{p/2}^{-1}$-fraction of each list suffices to let a single representation of the solution survive on expectation. This approach leads to an algorithm which runs in time $2^{0.05364n}$.

Our contribution. We propose to choose $|I_1| = |I_2| = \frac{p}{2} + \varepsilon$ for some $\varepsilon > 0$ such that $|I_1 \cap I_2| = \varepsilon$. So we allow for ε columns $\mathbf{q}_i$ that appear on both sides of identity (1). Thus every solution I is written as the symmetric difference $I = I_1 \Delta I_2 := I_1 \cup I_2 \setminus (I_1 \cap I_2)$, where we cancel out all ε elements in the intersection of I_1 and I_2.

Let us compare our approach with the realization of the search step in the algorithms of Stern [29] and MMT [22]. In Stern's algorithm both index sets I_1, I_2 are chosen in a disjoint fashion. Thus every solution I only has a unique representation as the union of I_1 and I_2. MMT choose fully intersecting sets I_1, I_2, but they only consider a union of *disjoint* sets I_1, I_2. Basically, this allows

that every of the p elements in $I = I_1 \cup I_2$ can appear either as an element of I_1 or as an element of I_2, so it can appear on both sides of identity (1).

In contrast, we choose fully intersecting sets I_1, I_2 and additionally allow for a union of *intersecting* sets. Thus, we additionally allow that even those $k + \ell - p$ elements that are *outside of* $I = I_1 \cup I_2$ may appear in I_1, I_2 as long as they appear in both sets, and thus cancel out. This drastically increases the number of representations, since for random code instances the number of zeros in an error vector $\mathbf{e}$ is much larger than the number of ones. Whereas MMT only allow to split each 1-entry of $\mathbf{e}$ into two parts, either $1 = 0 + 1$ or $1 = 1 + 0$, we also allow to split each 0-entry of $\mathbf{e}$ into two parts, either $0 = 0 + 0$ or $0 = 1 + 1$. Hence our benefit comes from using the equation $1 + 1 = 0$ in $\mathbb{F}_2$. Notice that our approach therefore increases the number of representation per solution I to $\binom{p}{p/2} \cdot \binom{k+\ell-p}{\epsilon}$.

Our main algorithmic task that we describe in this work is the construction of two lists $\mathcal{L}_1, \mathcal{L}_2$ such that a single representation of each solution survives. This is realized by a three-level divide-and-conquer algorithm that is similar to Wagner's generalized birthday algorithm [30].

Our enhanced representation technique allows us to significantly lower the asymptotic running time to $2^{0.04934n}$. The following figure shows the curve of the complexity coefficient for the two most recent algorithms [4, 22] compared to our new algorithm.

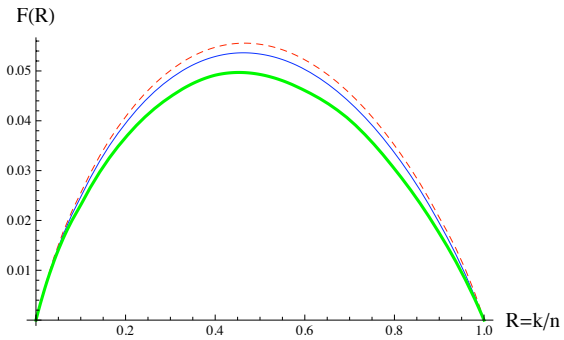

Fig. 1: Comparison of $F(R)$ for code rates $0 < R < 1$ for bounded distance decoding. Our algorithm is represented by the thick curve, MMT is the thin curve and Ball-collision is the dashed curve.

2 Generalized Information Set Decoding

We now give a detailed description of a generalized information set decoding (ISD) framework as described by Finiasz and Sendrier [11] in 2009. Re-

call that the input to an ISD algorithm is a tuple $(\mathbf{H}, \mathbf{s})$ where $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$ is a parity check matrix of a random linear $[n, k, d]$-code and $\mathbf{s} = \mathbf{He}$ is the syndrome of the unknown error vector $\mathbf{e}$ of weight $\omega := \mathrm{wt}(\mathbf{e}) = \lfloor \frac{d-1}{2} \rfloor$.

ISD is a randomized Las Vegas type algorithm that iterates two steps until the solution $\mathbf{e}$ is found. The first step is an initial linear transformation of the parity check matrix $\mathbf{H}$, followed by a search phase as the second step.

In the initial transformation, we permute the columns of $\mathbf{H}$ by multiplying with a random permutation matrix $\mathbf{P} \in \mathbb{F}_2^{n\times n}$. Then we perform Gaussian elimination on the rows of $\mathbf{HP}$ by multiplying with an invertible matrix $\mathbf{T} \in \mathbb{F}_2^{(n-k)\times(n-k)}$. This yields a parity check matrix $\tilde{\mathbf{H}} = \mathbf{THP}$ in quasi-systematic form containing a $\mathbf{0}$-submatrix in the right upper corner as illustrated in Fig. 2. Here we denote by $\mathbf{Q}^I$ the projection of $\mathbf{Q}$ to the rows defined by the index set $I \subset \{1, \ldots, n-k\}$. Analogously, we denote by $\mathbf{Q}_I$ the projection of $\mathbf{Q}$ to its columns. In particular we define $[\ell] := \{1, \ldots, \ell\}$ and $[\ell, n-k] = \{\ell, \ldots, n-k\}$. We denote the initial transformation $\mathsf{Init}(\mathbf{H}) := \mathbf{THP}$.

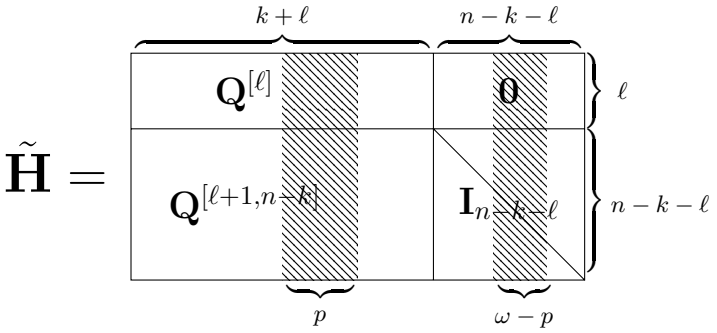

Fig. 2: Parity check matrix $\tilde{\mathbf{H}}$ in quasi-systematic form.

We set $\tilde{\mathbf{s}} := \mathbf{Ts}$ and look for an ISD-solution $\tilde{\mathbf{e}}$ of $(\tilde{\mathbf{H}}, \tilde{\mathbf{s}})$, i.e. we look for an $\tilde{\mathbf{e}}$ satisfying $\tilde{\mathbf{H}}\tilde{\mathbf{e}} = \tilde{\mathbf{s}}$ and $\mathrm{wt}(\tilde{\mathbf{e}}) = \omega$. This yields a solution $\mathbf{e} = \mathbf{P}\tilde{\mathbf{e}}$ for the original problem. Notice that applying the permutation matrix to $\tilde{\mathbf{e}}$ leaves the weight unchanged, i.e. $\mathrm{wt}(\mathbf{e}) = \omega$, and $\mathbf{THe} = \tilde{\mathbf{H}}\tilde{\mathbf{e}} = \tilde{\mathbf{s}} = \mathbf{Ts}$ implies $\mathbf{He} = \mathbf{s}$ as desired. In the search phase, we try to find all error vectors $\tilde{\mathbf{e}}$ that have a specific weight distribution, i.e. we search for vectors that can be decomposed into $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2) \in \mathbb{F}_2^{k+\ell} \times \mathbb{F}_2^{n-k-\ell}$ where $\mathrm{wt}(\tilde{\mathbf{e}}_1) = p$ and $\mathrm{wt}(\tilde{\mathbf{e}}_2) = \omega - p$. Since $\mathbf{P}$ shuffles $\mathbf{e}$'s coordinates into random positions, $\tilde{\mathbf{e}}$ has the above weight distribution with probability

$$\mathcal{P} = \frac{\binom{k+l}{p}\binom{n-k-l}{\omega-p}}{\binom{n}{\omega}} \quad . \tag{2}$$

The inverse probability $\mathcal{P}^{-1}$ is the expected number of repetitions until $\tilde{\mathbf{e}}$ has the desired distribution. Then it suffices to find the truncated vector $\tilde{\mathbf{e}}_1 \in \mathbb{F}_2^{k+\ell}$ that represents the position of the first p ones. To recover the full error vector $\tilde{\mathbf{e}} = (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2)$, the missing coordinates $\tilde{\mathbf{e}}_2$ are obtained as the last $n-k-\ell$ coordinates of $\mathbf{Q}\tilde{\mathbf{e}}_1 + \tilde{\mathbf{s}}$. Hence, the goal in the ISD search phase is to compute the truncated error vector $\tilde{\mathbf{e}}_1$ efficiently. For the computation of $\tilde{\mathbf{e}}_1$ we focus on the submatrix $\mathbf{Q}^{[\ell]} \in \mathbb{F}_2^{\ell \times (k+\ell)}$. Since we fixed the $\mathbf{0}$-submatrix in the right-hand part of $\tilde{\mathbf{H}}$, we ensure that $\mathbf{Q}\tilde{\mathbf{e}}_1$ exactly matches the syndrome $\tilde{\mathbf{s}}$ on its first ℓ coordinates. Finding an $\tilde{\mathbf{e}}_1$ with such a property was called the *submatrix matching problem* in [22].

Definition 1 (Submatrix Matching Problem). *Given a random matrix* $\mathbf{Q} \in_R \mathbb{F}_2^{\ell \times (k+\ell)}$ *and a target vector* $\mathbf{s} \in \mathbb{F}_2^{\ell}$*, the* submatrix matching problem (SMP) *consists in finding a set* I *of size* p *such that the corresponding columns of* $\mathbf{Q}$ *sum up to* $\mathbf{s}$*, i.e. to find* $I \subseteq [1, k+\ell]$*,* $|I| = p$ *such that*

$$\sigma(\mathbf{Q}_I) := \sum_{i \in I} \mathbf{q}_i = \mathbf{s}, \text{ where } \mathbf{q}_i \text{ is the } i\text{-th column of } \mathbf{Q}.$$

Note that the SMP itself can be seen as just another syndrome decoding instance with parity check matrix $\mathbf{Q}$, syndrome $\mathbf{s} \in \mathbb{F}_2^{\ell}$ and parameters $[k+\ell, \ell, p]$.

Our improvement stems from a new algorithm ColumnMatch allowing to solve the SMP more efficiently by using more representations of a solution I. In Alg. 1 we describe the resulting ISD algorithm. Here we denote for a vector $\mathbf{x} \in \mathbb{F}_2^n$ and an index set $I \subset [n]$ by $\mathbf{x}_I \in \mathbb{F}_2^{|I|}$ the restriction of $\mathbf{x}$ to the coordinates of I.

Algorithm 1 GeneralizedISD

Input: Parity check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$, syndrome $\mathbf{s} = \mathbf{H}\mathbf{e}$ with $\mathrm{wt}(\mathbf{e}) = \omega$.
Output: Error $\mathbf{e} \in \mathbb{F}_2^n$
Parameters: p, ℓ

Repeat
  Compute $\tilde{\mathbf{H}} \leftarrow \mathsf{Init}(\mathbf{H})$ and $\tilde{\mathbf{s}} \leftarrow \mathbf{T}\mathbf{s}$ where $\tilde{\mathbf{H}} = \mathbf{T}\mathbf{H}\mathbf{P}$, $\mathbf{P}$ random permutation.
  Compute $\mathcal{L} =$ ColumnMatch$(\mathbf{Q}^{[\ell]}, \tilde{\mathbf{s}}_{[\ell]}, p)$.
  For all solutions $\tilde{\mathbf{e}}_1 \in \mathcal{L}$ **do**
    If $\mathrm{wt}(\mathbf{Q}\tilde{\mathbf{e}}_1 + \tilde{\mathbf{s}}) = \omega - p$ **then**
      Compute $\widetilde{\mathbf{e}} \leftarrow (\tilde{\mathbf{e}}_1, \tilde{\mathbf{e}}_2) \in \mathbb{F}_2^n$ where $\tilde{\mathbf{e}}_2 \leftarrow (\mathbf{Q}\tilde{\mathbf{e}}_1 + \tilde{\mathbf{s}})_{[\ell+1, n-k]}$
      Output $\mathbf{e} = \widetilde{\mathbf{e}}\mathbf{P}$.

Let $T := T(n, R; p, \ell)$ denote the running time of ColumnMatch. Then the running time of GeneralizedISD is $\mathcal{P}^{-1} \cdot T$.

3 The Merge-Join Building Block

In order to realize our improved SMP algorithm, we first introduce an essential building block that realizes the following task. Given a matrix $\mathbf{Q} \in \mathbb{F}_2^{\ell \times (k+\ell)}$ and two lists $\mathcal{L}_1$ and $\mathcal{L}_2$ containing binary vectors $\mathbf{x}_1, \ldots, \mathbf{x}_{|\mathcal{L}_1|}$ and $\mathbf{y}_1, \ldots, \mathbf{y}_{|\mathcal{L}_2|}$ of length $k+\ell$, we aim to join those elements $\mathbf{x}_i$ and $\mathbf{y}_j$ into a new list $\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2$ whose sum has weight p, i.e. $\mathrm{wt}(\mathbf{x}_i + \mathbf{y}_j) = p$. Furthermore, we require that the corresponding column-sum of $\mathbf{Q}$ already matches a given target $\mathbf{t} \in \mathbb{F}_2^r$ on its right-most $r \leq \ell$ coordinates, i.e. $(\mathbf{Q}(\mathbf{x}_i + \mathbf{y}_j))_{[r]} = \mathbf{t}$.

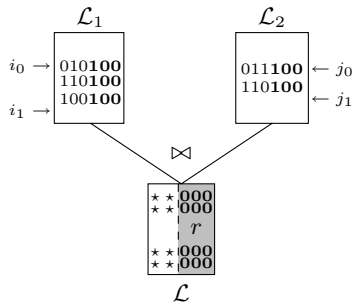

Fig. 3: Illustration of the MERGE-JOIN algorithm to obtain $\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2$.

Searching for matching vectors $(\mathbf{Q}\mathbf{y}_j)_{[r]} + \mathbf{t}$ and $(\mathbf{Q}\mathbf{x}_i)_{[r]}$ accomplishes this task. We call all matching vectors with weight different from p *inconsistent solutions*. Notice that we might also obtain the same vector sum from two different pairs of vectors from $\mathcal{L}_1, \mathcal{L}_2$. In this case we obtain a matched vector that we already have, which we call a *duplicate*. During our matching process we filter out all inconsistent solutions and duplicates.

The matching process is illustrated in Fig. 3. The complete algorithm is given as Alg. 2 and is based on a classical algorithm from Knuth [17] which realizes the collision search as follows. Sort the first list lexicographically according to the r-bit labels $L_1(\mathbf{x}_i) := (\mathbf{Q}\mathbf{x}_i)_{[r]}$ and the second list according to the labels $L_2(\mathbf{y}_j) := (\mathbf{Q}\mathbf{y}_j)_{[r]} + \mathbf{t}$. We add $\mathbf{t}$ to the labels of the second list to guarantee $(\mathbf{Q}(\mathbf{x}_i + \mathbf{y}_j))_{[r]} = \mathbf{t}$.

To detect all collisions, one now initializes two counters i and j starting at the beginning of the lists $\mathcal{L}_1$ and $\mathcal{L}_2$ and pointing at elements $\mathbf{x}_i$ and $\mathbf{y}_j$. As long as those elements do not yield a collision, either i or j is increased depending on the relative order of the labels $L_1(\mathbf{x}_i)$ and $L_2(\mathbf{y}_j)$. Once a collision $L_1(\mathbf{x}_i) = L_2(\mathbf{y}_j)$ occurs, four auxiliary counters i_0, i_1 and j_0, j_1 are initialized with i and j, respectively. Then i_1 and j_1 can further be incremented as long as the list elements retain the same labels, while i_0 and j_0 mark the first collision (i, j) between labels $L_1(\mathbf{x}_i)$ and $L_2(\mathbf{y}_j)$. Obviously, this procedure defines two

Algorithm 2 MERGE-JOIN

Input: $\mathcal{L}_1, \mathcal{L}_2, r, p$ and $\mathbf{t} \in \mathbb{F}_2^r$
Output: $\mathcal{L} = \mathcal{L}_1 \bowtie \mathcal{L}_2$

Lexicographically sort $\mathcal{L}_1$ and $\mathcal{L}_2$ according to the labels $L_1(\mathbf{x}_i) := (\mathbf{Q}\mathbf{x}_i)_{[r]}$ and $L_2(\mathbf{y}_j) := (\mathbf{Q}\mathbf{y}_j)_{[r]} + \mathbf{t}$.
Set collision counter $C \leftarrow 0$. Let $i \leftarrow 0$ and $j \leftarrow (|\mathcal{L}_2| - 1)$
While $i < |\mathcal{L}_1|$ and $j < |\mathcal{L}_2|$ **do**
  If $L_1(\mathbf{x}_i) <_{lex} L_2(\mathbf{y}_j)$ **then** $i++$
  If $L_1(\mathbf{x}_i) >_{lex} L_2(\mathbf{y}_j)$ **then** $j++$
  If $L_1(\mathbf{x}_i) = L_2(\mathbf{y}_j)$ **then**
    Let $i_0, i_1 \leftarrow i$ and $j_0, j_1 \leftarrow j$
    While $i_1 < |\mathcal{L}_1|$ and $L_1(\mathbf{x}_{i_1}) = L_1(\mathbf{x}_{i_0})$ **do** i_1++
    While $j_1 < |\mathcal{L}_2|$ and $L_2(\mathbf{y}_{j_1}) = L_2(\mathbf{y}_{j_0})$ **do** j_1++
    For $i \leftarrow i_0$ to $i_1 - 1$ **do**
      For $j \leftarrow j_0$ to $j_1 - 1$ **do**
        $C = C + 1$
        Insert collision $\mathbf{x}_i + \mathbf{y}_j$ into list $\mathcal{L}$ (unless filtered out)
    Let $i \leftarrow i_1$, $j \leftarrow j_1$
Output $\mathcal{L}, C$.

sets $C_1 = \{\mathbf{x}_{i_0}, \ldots, \mathbf{x}_{i_1}\}$ and $C_2 = \{\mathbf{y}_{j_0}, \ldots, \mathbf{y}_{j_1}\}$ such that all possible combinations yield a collision, i.e. the set $C_1 \times C_2$ can be added to the output list $\mathcal{L}$.

This procedure is then continued with $i \leftarrow i_1$ and $j \leftarrow j_1$ until one of the counters i, j arrives at the end of a list. As mentioned before, we remove on the fly inconsistent solutions with incorrect weight $\mathrm{wt}(\mathbf{x}_i + \mathbf{y}_j) \neq p$ and duplicate elements $\mathbf{x}_i + \mathbf{y}_j = \mathbf{x}_k + \mathbf{y}_\ell$.

Note that we introduced a collision counter C which allows us to take into account the time that is spent for removing inconsistent solutions and duplicates. The total running time of MERGE-JOIN is given by

$$T = \tilde{\mathcal{O}}\left(\max\left\{|\mathcal{L}_1|, |\mathcal{L}_2|, C\right\}\right) \quad .$$

Assuming uniformly distributed labels $L_1(\mathbf{x}_j)$ and $L_2(\mathbf{y}_j)$ it holds that $\mathbb{E}\left[C\right] = |\mathcal{L}_1| \cdot |\mathcal{L}_2| \cdot 2^{-r}$.

4 Our New Algorithm for Solving the Submatrix Matching Problem

As explained in Section 2, improving the submatrix matching problem (SMP) automatically improves information set decoding (ISD).

Our new SMP algorithm is inspired by using *extended representations* similar to Becker, Coron and Joux [2] for the subset sum problem.

In the MMT algorithm [22] a weight-p error vector $\mathbf{e} \in \mathbb{F}_2^{k+\ell}$ is written as the sum $\mathbf{e}_1 + \mathbf{e}_2$. However, MMT only allow that every 1-entry splits to either a 1-entry in $\mathbf{x}_1$ and a 0-entry in $\mathbf{x}_2$, or vice versa. If $\mathrm{wt}(\mathbf{e}_1) = \mathrm{wt}(\mathbf{e}_2) = \frac{p}{2}$ this allows for $\binom{p}{p/2}$ different representations as a sum of two vectors.

Our key observation is that we can also split the 0-entries of $\mathbf{e}$ into either $(0,0)$ or $(1,1)$. Hence if we choose $\mathrm{wt}(\mathbf{e}_1) = \mathrm{wt}(\mathbf{e}_2) = \frac{p}{2} + \varepsilon$ then we gain a factor of $\binom{k+\ell-p}{\varepsilon}$, namely the number of positions where we split as $(1,1)$. Notice that in all coding-based scenarios $\mathrm{wt}(\mathbf{e})$ is relatively small compared to k and n. Thus $\mathbf{e}$ contains many more zeros than ones, from which our new representation heavily profits.

To solve the SMP, we proceed as follows. Let $I \subset [k+\ell]$ be the index set of cardinality p with $\sigma(\mathbf{Q}_I) = \mathbf{s}$ that we want to find.

We represent I by two index sets I_1 and I_2 of cardinality $\frac{p}{2} + \varepsilon$ contained in the whole interval $[k+l]$ and require I_1 and I_2 to intersect in a fixed number of ε coordinates as illustrated in Fig. 4.

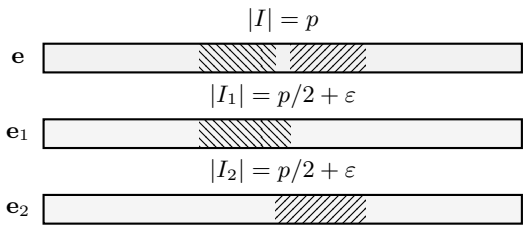

Fig. 4: Decomposition of an index set I into two overlapping index sets.

The resulting index set I is then represented as the symmetric difference $I_1 \Delta I_2 := (I_1 \cup I_2) \setminus (I_1 \cap I_2)$ which yields an index set I of cardinality p as long as I_1 and I_2 intersect in exactly ε positions.

It turns out that the optimal running time can be obtained by applying the representation technique twice, i.e. we introduce further representations of the index sets I_1 and I_2 on a second computation layer.

4.1 Our ColumnMatch Algorithm

Our algorithm can be described as a computation tree of depth three, see Fig. 7 for an illustration. We enumerate the layers from bottom to top, i.e. the third layer identifies the initial computation of disjoint base lists $\mathcal{B}_1$ and $\mathcal{B}_2$ and the zero layer identifies the final output list $\mathcal{L}$.

Recall that we aim to find an index set I of size p with $\sum_{i \in I} \mathbf{q}_i = \mathbf{s}$. We introduce parameters ε_1 and ε_2 representing the number of additional 1's we

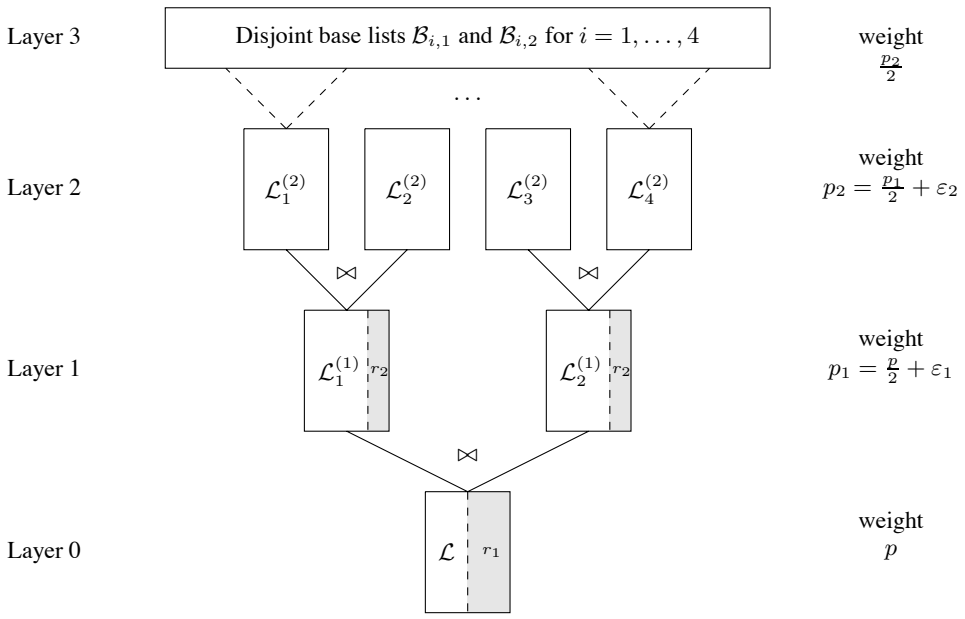

Fig. 5: Illustration of the COLUMNMATCH algorithm.

allow on the first and second layer, respectively. In the following description, we equip every object with an upper index that indicates its computation layer, e.g. a list $\mathcal{L}_j^{(2)}$ is contained in the second layer.

On the first layer, we search for index sets $I_1^{(1)}$ and $I_2^{(1)}$ in $[k + \ell]$ of size $p_1 := \frac{p}{2} + \varepsilon_1$ which intersect in exactly ε_1 coordinates such that $I = I_1^{(1)} \Delta I_2^{(1)}$. In other words, we create lists of binary vectors $\mathbf{e}_1^{(1)}$ and $\mathbf{e}_2^{(1)}$ of weight p_1 and search for tuples $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ such that $\mathrm{wt}(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = p$ and $\mathbf{Q}(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}) = \mathbf{s}$.

Note that the number of tuples $(\mathbf{e}_1^{(1)}, \mathbf{e}_2^{(1)})$ that represent a single solution vector $\mathbf{e}$ is

$$R_1(p, \ell; \varepsilon_1) := \binom{p}{\frac{p}{2}} \binom{k + l - p}{\varepsilon_1} \ . \tag{3}$$

To optimize the running time, we impose a constraint on $r_1 \approx \log_2 R_1$ coordinates of the corresponding vectors $\mathbf{Q}\mathbf{e}_i^{(1)}$ such that we can still expect to find one representation of the desired solution $\mathbf{e}$.

More precisely, the algorithm proceeds as follows. We first fix a random vector $\mathbf{t}_1^{(1)} \in_R \mathbb{F}_2^{r_1}$, set $\mathbf{t}_2^{(1)} := \mathbf{s}_{[r_1]} + \mathbf{t}_2^{(1)}$ and compute two lists

$$\mathcal{L}_i^{(1)} = \{\mathbf{e_i}^{(1)} \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(\mathbf{e_i}) = p_1 \text{ and } (\mathbf{Q}\mathbf{e}_i^{(1)})_{[r_1]} = \mathbf{t}_i^{(1)}\} \text{ for } i = 1, 2.$$

Observe that any two elements $\mathbf{e}_i^{(1)} \in \mathcal{L}_i^{(1)}$, $i = 1, 2$, already fulfill by construction the equation $(\mathbf{Q}(\mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}))_{[r_1]} = \mathbf{s}_{[r_1]}$, i.e. they already match the syndrome $\mathbf{s}$ on r_1 coordinates. In order to solve the SMP, we are interested in a solution $\mathbf{e} = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)}$ that matches the syndrome $\mathbf{s}$ on *all* ℓ positions and has weight *exactly* p. Once $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$ have been created, this can be accomplished by calling the MERGE-JOIN algorithm from Sect. 3 on input $\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)}$ with target $\mathbf{s}$, weight p and parameter ℓ.

It remains to show how to construct $\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)}$.

We represent $\mathbf{e}_i^{(1)}$ as a sum of two overlapping vectors $\mathbf{e}_{2i-1}^{(2)}, \mathbf{e}_{2i}^{(2)}$ both of weight $p_2 := \frac{p_1}{2} + \varepsilon_2$, i.e. we require the two vectors to intersect in exactly ε_2 coordinates. Altogether, the solution $\mathbf{e}$ is now decomposed as

$$\mathbf{e} = \mathbf{e}_1^{(1)} + \mathbf{e}_2^{(1)} = \mathbf{e}_1^{(2)} + \mathbf{e}_2^{(2)} + \mathbf{e}_3^{(2)} + \mathbf{e}_4^{(2)} \ .$$

Clearly, there are

$$R_2(p, \ell; \varepsilon_1, \varepsilon_2) = \binom{p_1}{p_1/2} \cdot \binom{k + \ell - p_1}{\varepsilon_2}$$

many representations for $\mathbf{e}_j^{(1)}$ where $p_1 = \frac{p}{2} + \varepsilon_1$. Similarly to the first layer, this allows us to fix $r_2 \approx \log R_2$ coordinates of the partial sums $\mathbf{Q}\mathbf{e}_i^{(2)}$ to some target values $\mathbf{t}_i^{(2)}$. More precisely, we draw two target vectors $\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)} \in \mathbb{F}_2^{r_2}$, set $\mathbf{t}_{2j}^{(2)} = (\mathbf{t}_j^{(1)})_{[r_2]} + \mathbf{t}_{2j-1}^{(2)}$ for $j = 1, 2$, and compute four lists

$$\mathcal{L}_i^{(2)} = \{\mathbf{e}_i^{(2)} \in \mathbb{F}_2^{k+l} \mid \mathrm{wt}(\mathbf{e}_i^{(2)}) = p_2 \text{ and } (\mathbf{Q}\mathbf{e}_i^{(2)})_{[r_2]} = \mathbf{t}_i^{(2)}\} \text{ for } i = 1, \ldots, 4.$$

Notice that by construction all combinations of two elements from either $\mathcal{L}_1^{(2)}, \mathcal{L}_2^{(2)}$ or $\mathcal{L}_3^{(2)}, \mathcal{L}_4^{(2)}$ match their respective target vector $\mathbf{t}_j^{(1)}$ on r_2 coordinates.

Creating the lists $\mathcal{L}_1^{(2)}, \ldots, \mathcal{L}_4^{(2)}$. We exemplary explain how to create $\mathcal{L}_1^{(2)}$. The remaining lists can be constructed analogously. We apply a classical Meet-in-the-middle collision search, i.e. we decompose $\mathbf{e}_1^{(2)}$ as $\mathbf{e}_1^{(2)} = \mathbf{y} + \mathbf{z}$ by two non-overlapping vectors $\mathbf{y}$ and $\mathbf{z}$ of length $k + \ell$. To be more precise, we first

choose a random partition of $[k+\ell]$ into two equal sized sets P_1 and P_2, i.e. $[k+\ell] = P_1 \cup P_2$ with $|P_1| = |P_2| = \frac{k+\ell}{2}$, and force $\mathbf{y}$ to have its $\frac{p_2}{2}$ 1-entries in P_1 and $\mathbf{z}$ to have its $\frac{p_2}{2}$ 1-entries in P_2. That is we construct two base lists

$$\mathcal{B}_1 := \{\mathbf{y} \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(\mathbf{y}) = \frac{p_2}{2} \text{ and } y_i = 0 \forall i \in P_2\}$$

and

$$\mathcal{B}_2 := \{\mathbf{z} \in \mathbb{F}_2^{k+\ell} \mid \mathrm{wt}(\mathbf{z}) = \frac{p_2}{2} \text{ and } z_i = 0 \forall i \in P_1\}.$$

We invoke MERGE-JOIN to compute $\mathcal{L}_1^{(2)} = \text{MERGE-JOIN}(\mathcal{B}_1, \mathcal{B}_2, r_2, p_2, \mathbf{t}_1^{(2)})$. Let $S_3 = |\mathcal{B}_1| = |\mathcal{B}_2|$ denote the size of the base lists and let C_3 be the total number of matched vectors that occur in MERGEJOIN (since the splitting is disjoint, neither duplicates nor inconsistencies can arise). Then MERGEJOIN needs time

$$T_3(p, \ell; \varepsilon_1, \varepsilon_2) = \mathcal{O}\left(\max\{S_3, C_3\}\right).$$

Clearly, we have

$$S_3 := S_3(p, \ell; \varepsilon_1, \varepsilon_2) = \binom{(k+\ell)/2}{p_2/2}.$$

Assuming uniformly distributed partial sums we obtain

$$\mathbb{E}\left[C_3\right] = \frac{S_3^2}{2^{r_2}} \ .$$

We would like to stress that decomposing $\mathbf{e}_1^{(2)}$ into $\mathbf{x}$ and $\mathbf{y}$ from disjoint sets P_1 and P_2 introduces a probability of loosing the vector $\mathbf{e}_1^{(2)}$ and hence the solution $\mathbf{e} = \mathbf{e}_1^{(2)} + \mathbf{e}_2^{(2)} + \mathbf{e}_3^{(2)} + \mathbf{e}_4^{(2)}$. For a randomly chosen partition P_1, P_2, the probability that $\mathbf{e}_1^{(2)}$ equally distributes its 1-entries over P_1 and P_2 is given by

$$\mathcal{P}_{\text{split}} = \frac{\binom{(k+\ell)/2}{p_2/2}^2}{\binom{k+\ell}{p_2}}$$

which is asymptotically inverse-polynomial in n. Choosing independent partitions $P_{i,1}, P_{i,2}$ and appropriate base lists $\mathcal{B}_{i,1}, \mathcal{B}_{i,2}$ for all four lists $\mathcal{L}_i^{(2)}$, we can guarantee *independent* splitting conditions for all the $\mathbf{e}_i^{(2)}$ yielding a total splitting probability of $\mathcal{P}_{\text{Split}} = (\mathcal{P}_{\text{split}})^4$ which is still inverse-polynomial in n.

After having created the lists $\mathcal{L}_i^{(2)}$, $i = 1, \ldots, 4$ on the second layer, two more applications of the MERGEJOIN algorithm suffice to compute the lists $\mathcal{L}_j^{(1)}$ on the first layer. Eventually, a last application of MERGEJOIN yields $\mathcal{L}$, whose entries are solutions to the SMP. See Alg. 3 for a complete pseudocode description.

Algorithm 3 COLUMNMATCH

Input: $\mathbf{Q} \in \mathbb{F}_2^{\ell \times k+\ell}, \mathbf{s} \in \mathbb{F}_2^{\ell}, p \leq k+\ell$
Output: List $\mathcal{L}$ of vectors in $\mathbf{e} \in \mathbb{F}_2^{k+\ell}$ with $\mathrm{wt}(\mathbf{e}) = p$ and $\mathbf{Qe} = \mathbf{s}$
Parameters: Choose optimal $\varepsilon_1, \varepsilon_2$ and set $p_1 = p/2 + \varepsilon_1$ and $p_2 = p_1/2 + \varepsilon_2$.

Choose random partitions $P_{i,1}, P_{i,2}$ of $[k+\ell]$ and create the base lists $\mathcal{B}_{i,1}$ and $\mathcal{B}_{i,2}$.
Choose $\mathbf{t}_1^{(1)} \in_R \mathbb{F}_2^{r_1}$ and set $\mathbf{t}_2^{(1)} = \mathbf{s}_{[r_1]} + \mathbf{t}_1^{(1)}$.
  Choose $\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)} \in_R \mathbb{F}_2^{r_2}$. Set $\mathbf{t}_2^{(2)} = (\mathbf{t}_1^{(1)})_{[r_2]} + \mathbf{t}_1^{(2)}$ and $\mathbf{t}_4^{(2)} = (\mathbf{t}_2^{(1)})_{[r_2]} + \mathbf{t}_3^{(2)}$.
  Compute $\mathcal{L}_i^{(2)} = \text{MERGE-JOIN}(\mathcal{B}_{i,1}, \mathcal{B}_{i,2}, r_2, p_2, \mathbf{t}_i^{(2)})$ for $i = 1, \ldots, 4$.
Compute $\mathcal{L}_i^{(1)} = \text{MERGE-JOIN}(\mathcal{L}_{2i-1}^{(2)}, \mathcal{L}_{2i}^{(2)}, r_1, p_1, \mathbf{t}_i^{(1)})$ for $i = 1, 2$.
Compute $\mathcal{L} = \text{MERGE-JOIN}(\mathcal{L}_1^{(1)}, \mathcal{L}_2^{(1)}, \ell, p, \mathbf{s})$.
Output $\mathcal{L}$.

It remains to estimate the complexity of COLUMNMATCH as a function of the parameters $(p, \ell; \varepsilon_1, \varepsilon_2)$, where $(\varepsilon_1, \varepsilon_2)$ are optimization parameters. Notice that the values r_i and p_i are fully determined by $(p, \ell; \varepsilon_1, \varepsilon_2)$. The base lists $\mathcal{B}_1$ and $\mathcal{B}_2$ are of size $S_3(p, \ell; \varepsilon_1, \varepsilon_2)$ as defined above.

The three consecutive calls to the MERGE-JOIN routine create lists $\mathcal{L}_j^{(2)}$ of size $S_2(p, \ell; \varepsilon_1, \varepsilon_2)$, lists $\mathcal{L}_i^{(1)}$ of size $S_1(p, \ell; \varepsilon_1, \varepsilon_2)$ and the final list $\mathcal{L}$ (which has not to be stored). More precisely, we obtain

$$S_i(p, \ell; \varepsilon_1, \varepsilon_2) = \mathbb{E}\left[|\mathcal{L}_j^{(i)}|\right] = \binom{k+\ell}{p_i} \cdot 2^{-r_i} \text{ for } i = 1, 2.$$

Here we assume uniformly distributed partial sums $\mathbf{Qe}_i^{(j)}$.

Let C_i for $i = 1, 2, 3$ denote the number of all matching vectors (including possible inconsistencies or duplicates) that occur in the three MERGE-JOIN steps. If we set $r_3 = 0$ and $r_0 = \ell$, then

$$\mathbb{E}[C_i] = S_i^2 \cdot 2^{r_i - r_{i-1}}.$$

Following the analysis of MERGE-JOIN in Sect. 3, the time complexities T_i of the three MERGE-JOIN steps is given by

$$T_i(p, \ell; \varepsilon_1, \varepsilon_2) = \max\{S_i, C_i\}.$$

The overall time and space complexity is thus given by

$$T(p, \ell; \varepsilon_1, \varepsilon_2) = \max\{T_3, T_2, T_1\} \tag{4}$$

and

$$S(p, \ell; \varepsilon_1, \varepsilon_2) = \max\{S_3, S_2, S_1\} \quad .$$

For optimizing $T(p, \ell; \varepsilon_1, \varepsilon_2)$ one has to compute the C_i. Heuristically, we can assume that the C_i achieve their expected values up to a constant factor. Since our heuristic analysis also relies on the fact that projected partial sums of the form $(\mathbf{Qe})_{[r]}$ yield uniformly distributed vectors in $\mathbb{F}_2^r$, a proper theoretical analysis needs to take care of a certain class of malformed input parity check matrices $\mathbf{H}$. We show how to obtain a provable variant of our algorithm that works for all but a negligible amount of input matrices $\mathbf{H}$ in App.A. The provable variant simply aborts computation if the C_i differ too much from their expectation.

5 Comparison of Asymptotic Complexity

We now show that we improve information set decoding by an exponential factor in comparison to the latest results [4, 22]. To compute the complexity coefficient $F(R)$ for our algorithm for a fixed code rate R, we need to optimize the parameters p, ℓ, ε_1 and ε_2 such that the expression

$$T(p, \ell; \varepsilon_1, \varepsilon_2) \cdot \mathcal{P}(p, \ell)^{-1} \tag{5}$$

is minimized under the natural constraints

$$\begin{aligned}
0 <& \ell < \min\{n-k, n-k-\omega-p\} \\
0 <& p < \min\{\omega, k+\ell\} \\
0 <& \varepsilon_1 < k + \ell - p \\
0 <& \varepsilon_2 < k + \ell - p_1 \\
0 <& R_2(p, \ell; \varepsilon_1, \varepsilon_2) < R_1(p, \ell; \varepsilon_1, \varepsilon_2) < \ell \ .
\end{aligned}$$

The time per iteration T is given by Eq. (4) and the number of iterations $\mathcal{P}^{-1}$ equals $\left(\binom{k+\ell}{p}\binom{n-k-\ell}{\omega-p}/\binom{n}{\omega}\right)^{-1}$ as given in Eq. (2).

For random linear codes, we can relate $R = k/n$ and $D = d/n$ via the Gilbert-Varshamov bound. Thus asymptotically we obtain $D = H^{-1}(1-R) + o(1)$, where H is the binary entropy function. For *bounded distance decoding*, we set $W := \omega/n = D/2$. We numerically determined the optimal parameters for several equidistant rates R and interpolated $F(R)$. To calculate $F(R)$ we make use of the well known approximation $\binom{\alpha n}{\beta n} = 2^{\alpha H(\beta/\alpha)n + o(n)}$. The results are shown in Fig. 1.

For *full decoding*, in the worst-case we need to decode a highest weight coset leader of the code C, its weight ω corresponds to the *covering radius* of C which is defined as the smallest radius r such that C can be covered by discrete balls of radius r. The Goblick bound [12] ensures that $r \geq nH^{-1}(1-R) + o(n)$ for *all* linear codes. Independently, Blinovskii [6] and Levitin [20] further proved

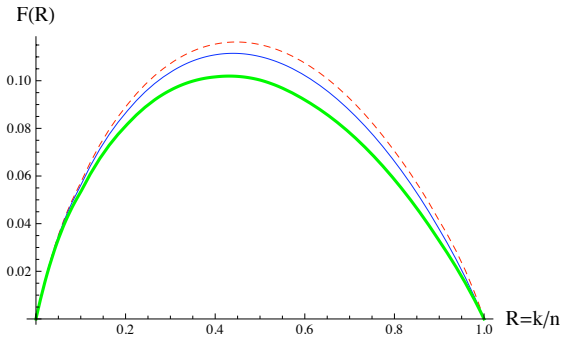

Fig. 6: $F(R)$ for full decoding. Our algorithm is represented by the thick curve, MMT is the thin curve and Ball-collision is the dashed curve.

that this bound is tight for *almost all* linear codes, i.e. $r = nH^{-1}(1-R)+o(n)$. This justifies our choice $W = H^{-1}(1-R)$ for the full decoding scenario.
We conclude by taking a closer look at the *worst-case* complexities of decoding algorithms for random linear codes and a typical McEliece setting with relative distance $D = 0.04$ and rate $R = 0.7577$. Notice that three out of the four parameter sets for security levels between 80 and 256 bit from [3] closely match these code parameters.

	half-dist.		full dec.		McEliece	
	time	space	time	space	time	space
Lee-Brickell	0.05752	-	0.1208	-	0.0857	-
Stern	0.05564	0.0135	0.1167	0.0318	0.0809	0.0327
Ball-collision	0.05559	0.0148	0.1164	0.0374	0.0807	0.0348
MMT	0.05364	0.0216	0.1116	0.0541	0.0760	0.0482
Our algorithm	0.04934	0.0286	0.1019	0.0769	0.0672	0.0586

Table 1: Comparison of worst-case complexity coefficients, e.g. the time columns represent the maximal complexity coefficient $F(R)$ for $0 < R < 1$.

All algorithms were optimized for speed, not for memory. For a comparison of full decoding with fixed memory, we can easily restrict Ball-collision, MMT and our new algorithm to the space complexity coefficient 0.0317 of Stern's algorithm which holds for $k \approx 0.446784$. In this case, we obtain time complexities $F_{\text{ball}}(R) = 0.1163$, $F_{\text{MMT}}(R) = 0.1129$ and $F_{\text{our}}(R) = 0.1110$, which shows that our improvement is not a pure time memory tradeoff.

For a better verifiability of our optimization and the resulting complexities, we make all data including the Mathematica code publicly available at

`http://cits.rub.de/personen/meurer.html`. If needed, this code may also be used to compute optimal parameters for arbitrary code parameters.

Acknowledgment. We would like to thank Dan Bernstein for several excellent comments, in particular he proposed to use random partitions for generating the base lists in the COLUMNMATCH algorithm.

References

1. M. Alekhnovich. More on Average Case vs Approximation Complexity. In *44th Symposium on Foundations of Computer Science (FOCS), pages 298–307, 2003*
2. A. Becker, J.-S. Coron, and A. Joux. Improved generic algorithms for hard knapsacks. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 364–385. Springer, 2011.
3. D.J. Bernstein, T. Lange and C. Peters. Attacking and Defending the McEliece Cryptosystem. In *Post-Quantum Cryptography, Second International Workshop, PQCrypto 2008*, pages 31-46. Springer, 2008.
4. D. J. Bernstein, T. Lange, and C. Peters. Smaller decoding exponents: ball-collision decoding. In *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 743–760. Springer, 2011.
5. R. J. M. Elwyn R. Berlekamp and H. C. van Tilborg. On the inherent intractability of certain coding problems. In *IEEE Transactions on Information Theory*, volume 24, pages 384–386, 1978.
6. V.M. Blinovskii. Lower asymptotic bound on the number of linear code words in a sphere of given radius in $\mathbb{F}_q^n$. In *Probl. Peredach. Inform., vol 23, pages 50-53, 1987*.
7. A. Canteaut and F. Chabaud. 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.
8. J.T. Coffey and R.M. Goodman. The complexity of information set decoding. In *IEEE Transactions on Information Theory*, volume 36, pages 1031-1037, 1990.
9. J.T. Coffey and R.M. Goodman. Any code of which we cannot think is good. In *IEEE Transactions on Information Theory*, volume 36, 1990.
10. J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich. A Distinguisher for High Rate McEliece Cryptosystems. In *YACC 2010*, full version available as eprint Report 2010/331, 2010.
11. M. Finiasz and N. Sendrier. Security bounds for the design of code-based cryptosystems. In M. Matsui, editor, *Asiacrypt 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 88–105. Springer, 2009.
12. T.J. Goblick, Jr. Coding for a discrete information source with a distortion measure. Ph.D. dissertation, Dept. of Elect. Eng., M.I.T., Cambridge, MA, 1962.
13. N.J. Hopper and M. Blum. Secure Human Identification Protocols. In *Lecture Notes in Computer Science*, volume 2248, Proceedings of *Advances in Cryptology - ASIACRYPT 2001*, pages 52-66. Springer 2001.
14. N. Howgrave-Graham and A. Joux. New generic algorithms for hard knapsacks. In *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 235–256. Springer, 2010.
15. J. P. Jordan. A variant of a public key cryptosystem based on goppa codes. *SIGACT News*, 15:61–66, January 1983.

16. E. Kiltz, K. Pietrzak, D. Cash, A. Jain and D. Venturi. Efficient Authentication from Hard Learning Problems. In *Advances in Cryptology - EUROCRYPT 2011*, pages 7-26. Springer, 2001.
17. D. Knuth. *Art of Computer Programming, Volume 3: Sorting and Searching*. Addison-Wesley Professional, 2 edition, 1998.
18. P. J. Lee and E. F. Brickell. An observation on the security of McEliece's public-key cryptosystem. In *Advances in Cryptology - EUROCRYPT 1988*, pages 275–280, 1988.
19. J. S. Leon. A probabilistic algorithm for computing minimum weights of large error-correcting codes. *IEEE Transactions on Information Theory*, 34(5):1354 – 1359, 1988.
20. L.B. Levitin. Covering radius of almost all linear codes satisfies the Goblick bound. In *IEEE Internat. Symp. on Information Theory*, Kobe, Japan, 1988.
21. R. J. McEliece. A public-key cryptosystem based on algebraic coding theory. In *Jet Propulsion Laboratory DSN Progress Report* 42–44, pages 114–116, 1978.
22. A. May, A. Meurer and E. Thomae. Decoding Random Linear Codes in $\tilde{\mathcal{O}}(2^{0.054n})$ In *Asiacrypt 2011*. Springer, 2011. *To appear.*
23. P. Q. Nguyen, I. E. Shparlinski, and J. Stern. Distribution of modular sums and the security of the server aided exponentiation. In *Progress in Computer Science and Applied Logic*, volume 20 of *Final proceedings of Cryptography and Computational Number Theory workshop, Singapore (1999)*, pages 331–224, 2001.
24. E. Prange. The Use of Information Sets in Decoding Cyclic Codes. *IRE Transaction on Information Theory*, volume 8, issue 5, pages 5-9, 1962.
25. C. Peters. Information-Set Decoding for Linear Codes over $\mathbb{F}_q$. In *Post-Quantum Cryptography, Third International Workshop, PQCrypto 2010*, pages 81-94. Springer, 2010.
26. O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual ACM Symposium on Theory of Computing (STOC)*, pages 84-93, 2005.
27. N. Sendrier. Finding the permutation between equivalent linear codes: The support splitting algorithm In *IEEE Transactions on Information Theory*, volume 46, pages 1193–1203, 2000.
28. N. Sendrier. On the security of the McEliece public-key cryptosystem. In M. Blaum, P. Farrell, and H. van Tilborg, editors, *Information, Coding and Mathematics*, pages 141–163. Kluwer, 2002. Proceedings of Workshop honoring Prof. Bob McEliece on his 60th birthday.
29. J. Stern. A method for finding codewords of small weight. In *Proceedings of the 3rd International Colloquium on Coding Theory and Applications*, pages 106–113, London, UK, 1989. Springer-Verlag.
30. D. Wagner. A generalized birthday problem. In *CRYPTO'2002*, pages 288–303, 2002.

A A Provable Variant of ColumnMatch

In order to obtain a provable variant of the ColumnMatch algorithm, we consider the following variant ProvableCM. Recover that one invocation of ColumnMatch requires the random choice of three target vectors $\mathbf{t}_1^{(1)} \in \mathbb{F}_2^{r_1}$ and $\mathbf{t}_1^{(2)}, \mathbf{t}_3^{(2)} \in \mathbb{F}_2^{r_2}$.

Essentially, in ProvableCM we repeatedly invoke ColumnMatch with different independent target values $\mathbf{t}_i^{(j)}$ and add some artificial abort criteria that prevent the lists in the computation from growing unexpectedly strong. More detailed, for an integer parameter $\Lambda \in \mathbb{N}$, we first choose random target values $\mathbf{t}_{1,i}^{(1)} \in \mathbb{F}_2^{(r_1)}$ and $\mathbf{t}_{1,j}^{(2)}, \mathbf{t}_{3,k}^{(2)} \in \mathbb{F}_2^{r_2}$ for $1 \leq i, j, k \leq 8\Lambda$ and invoke ColumnMatch$(\mathbf{t}_{1,i}^{(1)}, \mathbf{t}_{1,j}^{(2)}, \mathbf{t}_{3,j}^{(2)})$ for all $1 \leq i, j, k \leq 8\Lambda$ until a solution is found. Furthermore, every single computation is aborted as soon as a list exceeds its expected size by more than a factor of $2^{\gamma n}$ for a fixed constant $\gamma > 0$. We now aim to prove the following

Theorem 1. *For every $\gamma > 0$, the modified algorithm* ProvableCM *outputs a solution $\mathbf{e} \in \mathbb{F}_2^{k+\ell}$, i.e. $\mathbf{Qe} = \mathbf{s}$ and $\mathrm{wt}(\mathbf{e}) = p$, for a fraction of at least $1 - 60 \cdot 2^{-\gamma n}$ randomly chosen $\mathbf{Q} \in \mathbb{F}_2^{\ell \times (k+\ell)}$ with probability at least $1 - \frac{3}{e^2} > \frac{1}{2}$ in time $\tilde{\mathcal{O}}\left(T(p, \ell; \varepsilon_1, \varepsilon_2) \cdot 2^{3\gamma n}\right)$ where $T(p, \ell; \varepsilon_1, \varepsilon_2)$ is defined as in Eq.(4).*

We make use of the following helpful theorem that can be obtained by a straightforward modification of the result in [23, Theorem 3.2].

Theorem 2. *For a fixed matrix $\mathbf{Q} \in \mathbb{F}_2^{m \times n}$, a target vector $\mathbf{t} \in \mathbb{F}_2^m$ and an arbitrary set $\mathcal{B} \subset \mathbb{F}_2^n$, we define*

$$P_{\mathbf{Q}}(\mathcal{B}, \mathbf{t}) := \frac{1}{|\mathcal{B}|} \left|\{\mathbf{x} \in \mathcal{B} \ : \ \mathbf{Qx} = \mathbf{t}\}\right| \ .$$

Then for all $\mathcal{B} \subset \mathbb{F}_2^n$ it holds that

$$\frac{1}{2^{mn}} \sum_{\mathbf{Q} \in \mathbb{F}_2^{m \times n}} \sum_{\mathbf{t} \in \mathbb{F}_2^m} (P_{\mathbf{Q}}(\mathcal{B}, \mathbf{t}) - \frac{1}{2^m})^2 = \frac{2^m - 1}{2^m |\mathcal{B}|} \ .$$

The high-level idea for the proof of Theorem 1 is to consider the three different nodes of decomposition as illustrated in Fig. 7. For every such node, we introduce a random variable X_i indicating whether the algorithm fails at this point or not. The overall failure probability can then be upper bounded by using the union bound, i.e. $\mathbf{Pr}\left[\text{ProvableCM fails }\right] \leq \sum \mathbf{Pr}\left[X_i = 0\right]$. Hence, we need to upper bound the failure probability of every single node. For this purpose, we divide every X_i into three events X_i^j and set $X_i := \prod X_i^j$. We now define these events exemplarily for node X_1.

– X_1^1 represents the event that for at least one choice of the $\{\mathbf{t}_{1,j}^{(1)}\}$ the solution $\mathbf{e} \in \mathcal{L}$ has at least one representation $\mathbf{e} = \mathbf{e}_1 + \mathbf{e}_2$ with $(\mathbf{Q}\mathbf{e_1})_{[r_1]} = \mathbf{t}_{1,j}^{(1)}$ and $(\mathbf{Q}\mathbf{e_2})_{[r_1]} = \mathbf{s}_{[r_1]} + \mathbf{t}_{1,j}^{(1)}$.
– X_1^2 represents the event that for at least one choice of the $\{\mathbf{t}_{1,j}^{(1)}\}$ the size of lists $\mathcal{L}_1^{(1)}$ and $\mathcal{L}_2^{(1)}$ do not exceed the expected value by more than a factor of $2^{\gamma n}$.
– X_1^3 represents the event that for at least one choice of the $\{\mathbf{t}_{1,j}^{(1)}\}$ total number of collisions C_1, see Sect.4, does not exceed it's expected value by more than a factor of $2^{\gamma n}$.

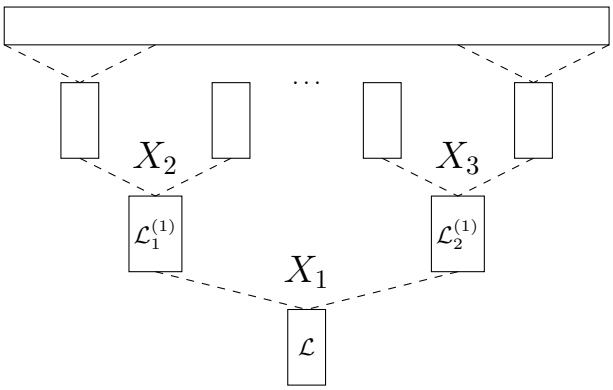

Fig. 7: Illustration of different decomposition nodes.

Basically, all these events depend on the structure of the matrix $\mathbf{Q}$ and we need to exclude some pathological cases yielding clustered distributions $\mathbf{Q}\mathbf{e}$ (for example the all-zero matrix). Hence, for all three events we define appropriate sets of "good" and "bad" matrices $\mathbf{Q}$ which eventually allow to upper bound the failure probabilities of these events. Applying Theorem 2 allows to upper bound the fraction of bad matrices. The following three lemmas control the amount of bad matrices $\mathbf{Q}$ and target values $\mathbf{t}_{i,j}^{(b)}$ for every X_i^j. We omit the proofs which can be adopted from the proof of [2, Theorem 2] in a straightforward way.

Lemma 1. *For all but a $\frac{1}{\Lambda-1}$ fraction of the $\mathbf{Q}$'s, the proportion of bad $\mathbf{t}'s$ w.r.t. to X_i^1 is smaller than $\frac{\Lambda-1}{\Lambda}$.*

Lemma 2. *For all but a $\frac{2\Lambda}{(\Lambda-1)^2}$ fraction of the $\mathbf{Q}$'s, the proportion of bad $\mathbf{t}'s$ w.r.t. to X_i^2 is smaller than $\frac{1}{2\Lambda}$.*

Lemma 3. *For all but a $\frac{16}{\Lambda}$ fraction of the $\mathbf{Q}$'s, the proportion of bad $\mathbf{t}'s$ w.r.t. to X_i^3 is smaller than $\frac{1}{4\Lambda}$.*

Using these lemmas, one can easily show that the total fraction of bad $\mathbf{Q}$'s for *one* of the three nodes can be bounded by

$$\frac{1}{\Lambda-1}+\frac{2\Lambda}{(\Lambda-1)^2}+\frac{16}{\Lambda}\leq\frac{20}{\Lambda}\ \text{ for } \Lambda\geq 7$$

and hence the total fraction of bad $\mathbf{Q}$'s for all three nodes is upper bounded by $\frac{60}{\Lambda}$. Furthermore, considering good $\mathbf{Q}$'s, the proportion of bad $\mathbf{t}$'s for *one* node is given by

$$\frac{\Lambda-1}{\Lambda}+\frac{1}{2\Lambda}+\frac{1}{4\Lambda}=1-\frac{1}{4\Lambda}$$

and hence we have

$$\mathbf{Pr}\left[X_i=0\right]=\mathbf{Pr}\left[\text{all } 8\Lambda \text{ many } \mathbf{t}\text{'s bad}\right]\leq\left(1-\frac{1}{4\Lambda}\right)^{8\Lambda}\leq e^{-2}\ .$$

Eventually this yields

$$\mathbf{Pr}\left[\text{PROVABLECM fails }\right]\leq\frac{3}{e^2}$$

for every good $\mathbf{Q}$ as stated in the theorem. Notice, that the worst-case running time of PROVABLECMis given by a total number of $\Lambda^3=2^{3\gamma n}$ invocations of COLUMNMATCH.