Robust Gray Codes Approaching the Optimal Rate

Roni Con* Dorsa Fathollahi† Ryan Gabrys‡ Mary Wootters§ Eitan Yaakobi¶

July 10, 2024

Abstract

Robust Gray codes were introduced by (Lolck and Pagh, SODA 2024). Informally, a robust Gray code is a (binary) Gray code $\mathcal{G}$ so that, given a *noisy* version of the encoding $\mathcal{G}(j)$ of an integer j, one can recover $\hat{j}$ that is close to j (with high probability over the noise). Such codes have found applications in differential privacy.

In this work, we present near-optimal constructions of robust Gray codes. In more detail, we construct a Gray code $\mathcal{G}$ of rate $1 - H_2(p) - \varepsilon$ that is efficiently encodable, and that is robust in the following sense. Supposed that $\mathcal{G}(j)$ is passed through the binary symmetric channel BSC_p with cross-over probability p, to obtain x. We present an efficient decoding algorithm that, given x, returns an estimate $\hat{j}$ so that $|j - \hat{j}|$ is small with high probability.

*Department of Computer Science, Technion - Israel Institute of Technology, Haifa, Israel, roni.con93@gmail.com.

†Department of Electrical Engineering, Stanford University, Stanford, CA, dorsafth@stanford.edu.

‡University of California San Diego, San Diego, CA, rgabrys@ucsd.edu.

§Department of Electrical Engineering, Stanford University, Stanford, CA, marykw@stanford.edu.

¶Department of Computer Science, Technion - Israel Institute of Technology, Haifa, Israel, yaakobi@cs.technion.ac.il

DF is partially supported by NSF grant CCF-2133154. The work of RG was partially supported by NSF Grant CCF-2212437. MW is partially supported by NSF grants CCF-2133154 and CCF-2231157. The work of RC and EY was supported by the European Union (DiDAX, 101115134). Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.
Part of this work was done while the authors were visiting the Simons Institute for the Theory of Computing.

1 Introduction

A *robust Gray code* is a Gray code that is robust to noise. In more detail, a robust Gray code $\mathcal{G}$ of length d is a map $\mathcal{G} : \{0, \ldots, N-1\} \to \{0,1\}^d$ so that:

- $\mathcal{G}$ is a Gray code: For all $j \in \{0, \ldots, N-1\}$, $\Delta(\mathcal{G}(j), \mathcal{G}(j+1)) = 1$, where Δ denotes the Hamming distance.[1]
- $\mathcal{G}$ is robust to noise from the *binary symmetric channel with cross-over probability* p (BSC_p), for some $p \in (0, 1/2)$: Let $\eta \sim \mathrm{Ber}(p)^d$ be a random noise vector. Then for any $j \in \{0, \ldots, N-1\}$, given $\mathcal{G}(j) \oplus \eta$, it should be possible to (efficiently) recover an estimate $\hat{j}$ so that $|j - \hat{j}|$ is small, with high probability over η.

As with standard error-correcting codes, we define the *rate* of a robust Gray code $\mathcal{G} : \{0, \ldots, N-1\} \to \{0,1\}^d$ by $\mathcal{R} = \frac{\log_2(N)}{d}$. The goal is then to make the rate as high as possible while achieving the above desiderata.

For intuition about the problem, consider two extreme examples. The first is the *unary code* of length $d = N$. The unary code simply encodes an integer j as j ones followed by $d-j$ zeros. It is not hard to see that if some random noise is introduced (with $p < 1/2$), it is possible to approximately identify j; it is the place where the bits go from being "mostly one" to "mostly zero." However, the rate of this code tends to zero very quickly; it has rate $\mathcal{R} = \log_2(d)/d$. The second extreme example is the classical Binary Reflected Code (BRC, see Definition 1). The BRC is a Gray code with $N = 2^d$ and hence rate $\mathcal{R} = 1$, as high as possible. However, the BRC is not at all robust. For example, the encodings of 0 and $N-1$ under the BRC differ by only a single bit, and more generally changing the "most significant bit" (or any highly significant bit) can change the value encoded by quite a lot. Our goal is something in between these extreme examples: A Gray code with good rate (as close to 1 as possible), but also with good robustness. Geometrically, one can think of this as a path that "fills up" as much of the Boolean cube $\{0,1\}^d$ as possible, while not getting too close to distant parts of itself too often.

Robust Gray codes were introduced by Lolck and Pagh in [LP24], motivated by applications to differential privacy. While their particular application (to differentially private histograms) is a bit involved, the basic idea is the following. In differential privacy, one adds noise to protect privacy, while hoping to still be able to estimate useful quantities about the data. Adding continuous noise (say, Laplace noise) to real values is standard, but it can be more practical to add noise from the BSC_p to binary vectors. This motivates a robust Gray code as a building block for differentially private mechanisms: It is a way of encoding integer-valued data into binary vectors, so that the original value can be estimated after noise from the BSC_p is added.

The original paper of Lolck and Pagh introduced a construction of robust Gray codes that transformed any binary error-correcting code $\mathcal{C}$ with rate $\mathcal{R}$ into a robust Gray code $\mathcal{G}$ with rate $\Omega(\mathcal{R})$. They showed that if $\mathcal{C}$ had good performance on BSC_p, then so did $\mathcal{G}$; more precisely, given $\mathcal{G}(j) \oplus \eta$, their decoder produced an estimate $\hat{j}$ so that

$$\Pr_{\eta}[|j - \hat{j}| \geq t] \leq \exp(-\Omega(t)) + \exp(-\Omega(d)) + O(P_{\mathrm{fail}}(\mathcal{C})),$$

[1] The paper [LP24] also gives a more general definition, where the code should have low *sensitivity*, meaning that $|\mathrm{Enc}_{\mathcal{G}}(j) - \mathrm{Enc}_{\mathcal{G}}(j+1)|$ is small; however, both their code and our code is a Gray code, so we specialize to that case (in which the sensitivity is 1).

where $P_{\text{fail}}(\mathcal{C})$ is the failure probability of $\mathcal{C}$ on the BSC_p. However, the constant in the $\Omega(R)$ in the rate in that work is at most $1/4$, which means that it is impossible for the construction of [LP24] to give a high-rate code, even if p is very small. The constant inside the term $\Omega(\mathcal{R})$ was improved to approach $1/2$ in [FW24].[2]

Our main result is a family of robust Gray codes that have rate approaching $1 - H_2(p)$ on the BSC_p, where $H_2(p) = -p\log_2(p) - (1-p)\log_2(1-p)$ is the binary entropy function. In fact, we prove a more general result, which takes any binary linear code $\mathcal{C}_{\text{in}}$ of rate $\mathcal{R}$, and transforms it into a robust Gray code $\mathcal{G}$ with rate approaching $\mathcal{R}$. This more general result is stated in Theorem 1 below; we instantiate it in Corollary 1 to achieve rate approaching $1 - H_2(p)$.

Theorem 1. Fix constants $p \in (0, 1/2)$ and a sufficiently small $\varepsilon > 0$. Fix a constant $\mathcal{R} \in (0,1)$. Let d be sufficiently large, in terms of these constants. Then there is an $n' = \Theta(\log d)$ so that the following holds. Suppose that there exists a binary linear $[n', k']_2$ code $\mathcal{C}_{\text{in}}$ with rate $k'/n' = \mathcal{R}$ so that $\mathcal{C}_{\text{in}}$ has a decoding algorithm $\text{Dec}_{\mathcal{C}_{\text{in}}}$ that has block failure probability on the BSC_p that tends to zero as $n' \to \infty$.[3] Then there is a robust Gray code $\mathcal{G} : \{0, 1, \dots N-1\} \to \mathbb{F}_2^d$ and a decoding algorithm $\text{Dec}_{\mathcal{G}} : \mathbb{F}_2^d \to \{0, 1, \dots N-1\}$ so that:

1. The rate of $\mathcal{G}$ is $\mathcal{R} - \varepsilon$.
2. Fix $j \in \{0, 1, \dots N-1\}$, let $\eta \sim \text{Ber}(p)^d$ be a random error vector, and let $\hat{j} := \text{Dec}_{\mathcal{G}}(\mathcal{G}(j) \oplus \eta)$, where $\eta \sim \text{Ber}(p)^d$. Then
$$\Pr_{\eta}[|j - \hat{j}| \geq t] \leq \exp(-\Omega(t)) + \exp\left(-\Omega\left(\frac{d}{\log d}\right)\right) ,$$
where the constants inside the $\Omega(\cdot)$ notation depend on p, ε, and $\mathcal{R}$.
3. The running time of $\mathcal{G}$ (the encoding algorithm) is $\tilde{O}(d^3)$ and the running time of $\text{Dec}_{\mathcal{G}}$ (the decoding algorithm) is $\tilde{O}(d^2)$ where the $\tilde{O}(\cdot)$ notation hides logarithmic factors.

Remark 1 (The running time of $\text{Dec}_{\mathcal{C}_{\text{in}}}$)**.** We note that the running time of $\text{Dec}_{\mathcal{C}_{\text{in}}}$ does not appear in Theorem 1. The reason is that for any code, the brute-force maximum-likelihood decoder runs in time $\text{poly}(n') \cdot 2^{k'}$. In the proof of Theorem 1, we will choose $k' = \log(n+1) \leq \log d$, which implies that $n' = O(\log d)$. Thus, the running time of $\text{Dec}_{\mathcal{C}_{\text{in}}}$ is at most $d \cdot \text{polylog(d)}$, and this is sufficiently small to obtain the bound on the running time of $\text{Dec}_{\mathcal{G}}$ in Theorem 1.

For the best quantitative results, we instantiate Theorem 1 by choosing $\mathcal{C}_{\text{in}}$ to be a binary code that achieves capacity on the binary symmetric channel, for example, polar codes [Ari08, TV13, GX14, HAU14, GRY20, BGN+22]; Reed-Muller codes [AS23, RP23]); or even a random linear code. This yields the following corollary.

Corollary 1. Let $p \in (0, 1/2)$, $\epsilon > 0$ be sufficiently small, and fix positive integers N and d sufficiently large, and with $\mathcal{R} := \frac{\log_2(N)}{d} = 1 - H_2(p) - \epsilon$. Then there is an efficiently encodable robust Gray code $\mathcal{G} : [N] \to \mathbb{F}_2^d$ of rate $\mathcal{R}$, so that the following holds. There is a polynomial-time algorithm $\text{Dec}_{\mathcal{G}} : \mathbb{F}_2^d \to [N]$ so that for any $j \in [N]$, for $\eta \sim \text{Ber}(p)^d$, $\hat{j} = \text{Dec}_{\mathcal{G}}(\mathcal{G}(j) \oplus \eta)$ satisfies
$$\Pr_{\eta}[|j - \hat{j}| \geq t] \leq \exp(-\Omega(t)) + \exp\left(-\Omega\left(\frac{d}{\log d}\right)\right) .$$

[2] The work [FW24] is by a subset of the authors of the current paper; we view it as a preliminary version of this work.

[3] See Definition 4 for a formal definition of the failure probability.

for any $t \geq 0$.

We note that $1 - H_2(p)$ is the Shannon capacity for BSC_p, which implies that the limiting rate of $1 - H_2(p)$ in Corollary 1 is optimal in the following sense.

Observation 1 (Optimality of Corollary 1)**.** Suppose that $\mathcal{G} : [N] \to \{0,1\}^d$ is a robust Gray code with rate $\mathcal{R} = 1 - H_2(p) + \theta$ for some constant $\theta > 0$. Let $\eta \sim \mathrm{Ber}(p)^d$ for some $p \in (0, 1/2)$. Then, for any procedure that recovers $\hat{j}$ from $\mathcal{G}(j) \oplus \eta$ and any $t = \Theta(1)$ and for sufficiently large N, we have $\Pr_\eta[|j - \hat{j}| > t] \geq 0.99$.

Proof. Suppose that $\mathcal{G}$ is in the statement of the observation, but that $\Pr_\eta[|j - \hat{j}| > t] < 0.99$. Then one could use $\mathcal{G}$ to communicate with non-negligible failure probability on the BSC_p as follows. The sender will encode a message $j \in \{0, \ldots, N-1\}$ as $\mathcal{G}(j)$ and send it over the channel. The receiver sees $\mathcal{G}(j) \oplus \eta$ and uses $\mathcal{G}$'s decoding algorithm (possibly inefficiently) to recover $\hat{j}$. Then the receiver returns $\tilde{j}$ chosen uniformly at random from the interval $I = \{\hat{j} - t, \hat{j} - t + 1, \ldots, \hat{j} + t\}$. The success probability of this procedure will be at least $0.01 \cdot \frac{1}{2t+1}$. Indeed, with probability at least 0.01, we have that $|j - \hat{j}| \leq t$ and hence $j \in I$, and, if that occurs, then with probability at least $1/(2t+1)$ we will have $\tilde{j} = j$, as $|I| = 2t + 1$. However, the converse to Shannon's channel coding theorem implies that the success probability for any code with rate $\mathcal{R}$ can be at most $\exp(-\Omega_{\theta,p}(d))$ (see, e.g., [SK20, Theorem 1.5]). This is a contradiction for sufficiently large d when $t = \Theta(1)$ is a constant (or even polynomial in d). □

1.1 Related Work

As mentioned earlier, robust Gray codes were originally motivated by applications in differential privacy, and have been used in that context; see [LP24, ALP21, ALS23, ACL+21] for more details on the connection. Beyond the initial construction of [LP24], the only prior work we are aware of is that of [FW24] mentioned above, which we build on in this paper. It is worth mentioning that there exist non-binary codes based on the Chinese Remainder Theorem [XXW20, WX10] that have nontrivial sensitivity, but in our work, we focus on binary codes.

Independent Work. While this paper was in preparation, it came to our attention that Guruswami and Wang have achieved similar results, but with different techniques [GW24]. In particular, their approach does not use code concatenation.

1.2 Technical Overview

Before diving into the details, we give an overview of our construction along with a discussion of how our approach leverages (and also departs from) ideas presented in previous work. In [LP24], the main idea was to transform a linear binary "Base" code $\mathcal{C}_B$ with rate R into a robust gray code $\mathcal{C}_\mathcal{G}$ with rate $\Omega(R)$. The technique used involves first concatenating four copies of a codeword from $\mathcal{C}_B$, of which two are bit-wise negated, in addition to some padding bits to form a codeword in an intermediate code, denoted $\mathcal{W}$, that is eventually transformed into the code $\mathcal{C}_\mathcal{G}$. Since each codeword in $\mathcal{W}$ (and also $\mathcal{C}_\mathcal{G}$) is composed of four copies of $x \in \mathcal{C}_B$, it is possible, even in the presence of noise, to allow one of the copies of x to be unrecoverable and still be able to use majority logic on the other three copies to determine the value of the encoded information.

In our preliminary version of this paper [FW24], we were able to use an ordering of $\mathcal{W}$, itself based on a Gray code, that allows us to construct each codeword in $\mathcal{W}$ using only *two* copies of a given codeword from $\mathcal{C}_B$, establishing that the rate $R/2$ is achievable. Under this setup, the ith codeword in $\mathcal{W}$ had the following format:

$$b_i \circ c_i \circ b_i \circ c_i \circ b_i,$$

where $c_i \in \mathcal{C}_B$ and where the b_i is a short padding sequence. However, it remained an open problem to determine whether it is possible to develop a general technique that converts a base code $\mathcal{C}_B$ of rate R to a robust gray code whose rate also approaches R. In this work, we provide an affirmative answer to the previous question. In order to develop such a technique, we rely on two simple ideas. The first idea is to define our base code $\mathcal{C}_B$ to be a concatenated coding scheme whose resulting code has certain performance guarantees on the BSC_p. The second idea has to do with the use of the padding bits. Rather than place our padding bits b_i in between different copies of $c_i \in \mathcal{C}_B$ to constuct codewords from $\mathcal{W}$, we will instead embed the markers b_i at regularly spaced intervals within c_i. Both these ideas will be discussed in more details in the following exposition. The full technical details of the construction are included in Section 2.

Before we get into a more detailed overview, we define the ingredients we will need. We require two codes $\mathcal{C}_{\mathrm{out}}$ and $\mathcal{C}_{\mathrm{in}}$ that are compatible under a concatenated error-correcting code scheme, meaning that the parameters are such that the concatenated code $\mathcal{C} = \mathcal{C}_{\mathrm{out}} \circ \mathcal{C}_{\mathrm{in}}$ makes sense. We will choose the outer code $\mathcal{C}_{\mathrm{out}} \subseteq \mathbb{F}_q^n$ to be high-rate linear $[n,k]_q$ code, which can correct a small fraction of worst-case errors; and as in Corollary 1, we will choose the inner code $\mathcal{C}_{\mathrm{in}}$ to be any binary code that achieves capacity on the BSC_p.

"Interpolating" between codewords of an intermediate code. We follow the same high-level idea as in [LP24, FW24], in that we first construct an *intermediate code* $\mathcal{W}$. The code $\mathcal{W}$ is a binary code constructed from $\mathcal{C}_{\mathrm{out}}$ and $\mathcal{C}_{\mathrm{in}}$, along with some bookkeeping information; we will describe it in the next paragraph. We will define an ordering $w_0, w_1, w_2, \ldots$ on the codewords of $\mathcal{W}$. Then we will create our final code $\mathcal{G}$ by "interpolating" between the codewords of $\mathcal{W}$, in order. We begin by defining $\mathcal{G}(0) = w_0$. Now, suppose that $z \in [d]$ is the first location that w_0 and w_1 differ; we define $\mathcal{G}(1)$ by flipping that bit in w_0. We continue in this way, flipping bits to interpolate between w_0 and w_1, and then between w_1 and w_2, and so on. We will choose parameters so that this will generate distinct encodings for each of our N codewords in the resulting Gray code.

Defining the intermediate code. While the high-level approach is similar to that in [LP24], as discussed in the beginning of this section, the improvements come from the definition of the intermediate code $\mathcal{W}$. We define it formally in Definition 5, but here we give some intuition for the construction. We begin with an ordering on the codewords $c_0, c_1, \ldots, c_{|\mathcal{C}_{\mathrm{out}}|}$ of the concatenated code $\mathcal{C} \subseteq \mathbb{F}_2^{n'n}$. This ordering (formally defined in Section 2.2) has the property that to get from the codeword c_{i-1} to the codeword c_i, one must simply add one row of the generator matrix A of $\mathcal{C}$.

Now, to construct the ith codeword w_i in $\mathcal{W}$, we proceed as follows. Let $b_i \in \{0,1\}$ be 0 if i is even and 1 if i is odd, and let $\vec{b_i}$ denote the bit b_i repeated many times.[4] Because of our ordering

[4]The number of times it is repeated is B, the distance of the inner code. Since the inner code has short length, $\vec{b_i}$ is also not very long, relative to n.

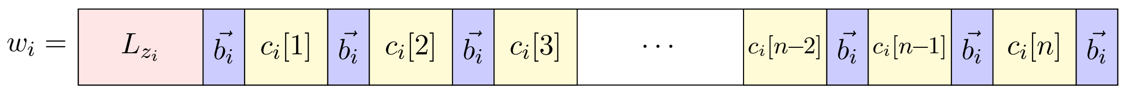
on $\mathcal{C}$, the only information we need to describe how to transition from c_{i-1} to c_i is the index of which row of A we must add; call this index $z_i \in [kk']$. Let L_{z_i} denote an encoding under the repetition code of this information z_i; since z_i is short, L_{z_i} can still be fairly short and also be extremely robust against the BSC_p. Consider a codeword $c_i \in \mathcal{C}_{\text{out}} \circ \mathcal{C}_{\text{in}}$. This codeword begins with a codeword $\sigma_i \in \mathcal{C}_{\text{out}}$, and has the form

$$c_i = c_i[1] \circ c_i[2] \circ \cdots \circ c_i[n],$$

where $\circ$ denotes concatenation and where for all $m \in \{1, \ldots, n\}$, we have

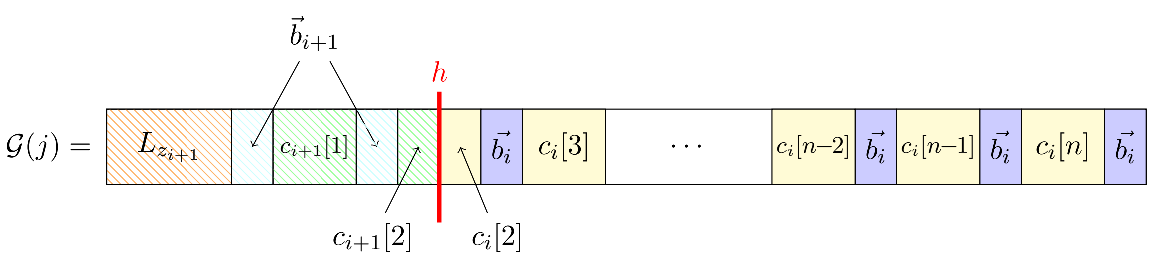
$$c_i[m] = \mathcal{C}_{\text{in}}(\sigma_i[m]) \in \mathcal{C}_{\text{in}}.$$

We will arrange these inner codewords $c_i[m] \in \mathcal{C}_{\text{in}}$ along with the quantities L_{z_i} and b_i in the following way:



That is, we alternate the inner codewords $c_i[m]$ with bursts of the bit b_i, and then include L_{z_i} at the beginning.

Decoding the resulting robust Gray code. To see why we define the intermediate code like we do, let us consider what a codeword $\mathcal{G}(j)$ of our robust Gray code looks like. Suppose that $\mathcal{G}(j)$ was an interpolation between w_i and w_{i+1}. Thus, for some "crossover point" $h \in [d]$, $\mathcal{G}(j)$ might look like this:



That is, everything before the "crossover point" h has been changed from w_i to w_{i+1}. This picture gives us some intuition for how we should decode $\mathcal{G}(j) \oplus \eta$ in order to obtain an estimate for j.

The high-level steps in this case would be:

- **Identify the approximate location of h.** Observe that the bit b_i is the opposite of the bit b_{i+1}. Thus, with high probability, we can look at the chunks of $\mathcal{G}(j) \oplus \eta$ that contain the b's and choose a point where they appear to "switch over" as an approximation of h.

- **Decode $\mathcal{C}_{\text{in}}$.** Next, on each chunk that is either $c_{i+1}[r]$ or $c_i[r]$, we run the decoder for $\mathcal{C}_{\text{in}}$ to correctly decode most of them. This gives us correct estimates for most of the $\sigma_{i+1}[r]$ or $\sigma_i[r]$.

- **Recover a noisy version of $\sigma_i \in \mathcal{C}_{\mathbf{out}}$.** Recall that L_{i+1} contains all the information necessary to recover c_i from c_{i+1} and vice versa. Thus, after decoding L_{i+1}, we can convert all of the $\sigma_{i+1}[r]$'s (at least, those which we have correctly recovered and which we have correctly identified as belonging to w_{i+1} using our estimate of h) into $\sigma_i[r]$ for all $r \in [n]$.

- **Decode $\mathcal{C}_{\mathbf{out}}$ to obtain i.** Given our noisy estimates of $\sigma_i[r]$ for all $r \in [n]$, we can now run the decoding algorithm of $\mathcal{C}_{\text{out}}$. Recall that $\mathcal{C}_{\text{out}}$ can handle a small fraction of worst-case errors; we will show that indeed our estimates of $\sigma_i[r]$ are incorrect for only a small fraction of r's. After correctly decoding, we can recover i.[5]

- **Recover $\hat{j}$.** Having correctly identified i and approximately identified h (with high probability), we can now estimate j, which is a function only of i and h.

Of course, there are many more details to be accounted for. First, one must of course work out the probability of success of all of the above steps, and work out the parameters. Second, there are several corner cases not captured in the picture above, depending on where the crossover point h lands. In the rest of the paper, we tackle these details. In Section 2, we formally define our construction; in Section 3 we state our decoding algorithm; and in Section 4 we analyze it and prove Theorem 1.

2 Definitions and Construction

2.1 Notation and useful definitions

We begin with some notation. For two vectors x, y, we use $\Delta(x, y)$ to denote the Hamming distance between x and y, and we use $\|x\|$ to denote the Hamming weight of x (that is, the number of non-zero coordinates). For an integer n, we use $[n]$ to denote the set $\{1, \ldots, n\}$. For two strings or vectors, u, and v we denote by $u \circ v$ their concatenation. Throughout this paper, we shall move freely between representation of vectors as strings and vice versa. For a string u, we define $\text{pref}_m(u)$ to be the prefix of u of length m and similarly $\text{suff}_m(u)$ will denote the last m symbols of u. For a vector v and an integer $i \geq 1$, we typically use $v[i]$ to denote the ith entry of v; one exception, defined formally below, is that for a codeword c in the concatenated code $\mathcal{C}_{\text{out}} \circ \mathcal{C}_{\text{in}}$ and for $m \in [n]$, $c[m] \in \mathcal{C}_{\text{in}}$ refers to the mth inner codeword in c.

We will use the following versions of the Chernoff/Heoffding bounds.

Lemma 1 (Multiplicative Chernoff bound; see, e.g., [MU17])**.** Suppose $X_1, \ldots, X_n$ are independent identically distributed random variables taking values in $\{0, 1\}$. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$. Then, for any $0 < \alpha < 1$:

$$\Pr[X > (1+\alpha)\mu] < e^{-\frac{\mu\alpha^2}{3}}$$

and

$$\Pr[X < (1-\alpha)\mu] < e^{-\frac{\mu\alpha^2}{2}}$$

Lemma 2 (Hoeffding's Inequality; see, e.g., [MU17])**.** Suppose $X_1, \ldots, X_n$ are independent random variables (not necessarily identically distributed) taking values in ± 1. Let $X = \sum_{i=1}^n X_i$ and $\mu = \mathbb{E}[X]$. Then for any $t \geq 0$,

$$\Pr[|X - \mu| \geq t] \leq 2\exp(-t^2/2n).$$

[5]In order to recover i efficiently, we leverage the particular ordering that we used on the codewords of $\mathcal{C}_{\text{out}}$.

Gray codes were introduced in [Gra53] (see also, e.g., [Knu11]) which also defined a particular Gray code called the *binary reflected code*. We will use this Gray code to order the codewords in one of our ingredient codes.

Definition 1 (Binary Reflected Code, [Gra53])**.** Let k be a positive integer. The **Binary Reflected Code (BRC)** is a map $\mathcal{R}_k : \{0, \ldots, 2^k - 1\} \to \mathbb{F}_2^k$ defined recursively as follows.

1. For $k = 1$, $\mathcal{R}_1(0) = 0$ and $\mathcal{R}_1(1) = 1$.
2. For $k > 1$, for any $i \in \{0, \ldots, 2^k - 1\}$,

$$\mathcal{R}_k(i) = \begin{cases} \mathcal{R}_{k-1}(i) \circ 0 & i < 2^{k-1} \\ \mathcal{R}_{k-1}(2^k - i - 1) \circ 1 & i \geq 2^{k-1} \end{cases}$$

Before continuing, we introduce some a few more definitions related to the BRC.

Definition 2. For $i \in \{1, 2, \ldots, 2^k - 1\}$, let z_i be the unique index where

$$\mathcal{R}_k(i)[z_i] \neq \mathcal{R}_k(i-1)[z_i].$$

Let $\mathcal{N}_k(z, i)$ be the number of $t \in \{0, 1, \ldots, i\}$ so that $z_t = z$.

That is, the value z_i is the index on which the ith codeword in the BRC differs from the previous codeword; equivalently, z_i is the integer for which $\mathcal{R}_k(i) = \mathcal{R}_k(i-1) + e_{z_i}$ where e_{z_i} is the z_ith unit vector. $\mathcal{N}_k(z, i)$ counts the number of codewords among $\{\mathcal{R}_k(0), \ldots, \mathcal{R}_k(i)\}$ that differ from the previous codeword in the zth index. We give an example of all of this notation below in Example 1.

Example 1. To illustrate Definitions 1 and 2, we give an example for $k = 1, 2, 3$. For $k = 1$, we have:

i	0	1
$\mathcal{R}_1(i)$	0	1

For $k = 2$, we have:

i	0	1	2	3
$\mathcal{R}_2(i)[0]$	0	1	1	0
$\mathcal{R}_2(i)[1]$	0	0	1	1

For $k = 3$, we have:

i	0	1	2	3	4	5	6	7
$\mathcal{R}_3(i)[0]$	0	1	1	0	0	1	1	0
$\mathcal{R}_3(i)[1]$	0	0	1	1	1	1	0	0
$\mathcal{R}_3(i)[2]$	0	0	0	0	1	1	1	1

The pattern is that in order to obtain the table for $\mathcal{R}_k$, we take the table for $\mathcal{R}_{k-1}$, and repeat it two times, first forwards and then backwards; then we add $\mathbf{0} \circ \mathbf{1}$ as the final row.

Next we give some examples of z_i and $\mathcal{N}_k$. For $k = 3$, we have the following values of z_i:

i	1	2	3	4	5	6	7
z_i	0	1	0	2	0	1	0

That is, $\mathcal{R}_3(0) = (0,0,0)$ and $\mathcal{R}_3(1) = (1,0,0)$ differ in the $z_1 = 0$ component, $\mathcal{R}_3(1) = (1,0,0)$ and $\mathcal{R}_3(2) = (1,1,0)$ differ in the $z_2 = 1$ component, and so on. Then, for example, $\mathcal{N}_{k=3}(z = 0, i = 3) = 2$, as there are two values of $t \leq i$ (names, $t - 1$ and $t = 3$) so that $z_t = 2$. As a few more examples, we have $\mathcal{N}_{k=3}(z = 1, i = 3) = 1$, and $\mathcal{N}_{k=3}(z = 0, i = 7) = 4$.

Below in Observation 2, we state a few useful facts about the z_i and $\mathcal{N}_k(z, i)$. Briefly, the reason these facts are useful for us is that we will use $\mathcal{R}_k$ to create the ordering on the codewords $c_i \in \mathcal{C}$ and $w_i \in \mathcal{W}$ discussed in the introduction. Understanding z_i and $\mathcal{N}_k(z, i)$ will be useful for efficiently computing indices in this ordering.

Observation 2 (Bit Flip Sequence of BRC)**.** For $k \geq 1$, the following holds:

1. The index z_i is equal to zero if and only if i is odd.
2. $\mathcal{N}_k(z, i) = \left\lfloor \frac{i+2^z}{2^{z+1}} \right\rfloor$.

Proof. Let $Z_k = (z_1, z_2, \ldots, z_{2^k-1})$, where the z_t's are defined with respect to k, as in the statement of the observation. We first observe that for any $k \geq 2$,

$$Z_k = Z_{k-1} \circ (k-1) \circ Z_{k-1}. \tag{1}$$

Indeed, for the base case $k = 2$, this follows by inspection: We have $Z_1 = 0$, and $Z_2 = 0, 1, 0$. For $k > 2$, it is clear from construction that $Z_k = Z_{k-1} \circ (k-1) \circ \overleftarrow{Z_{k-1}}$, where the $\overleftarrow{\cdot}$ notation means that the sequence is reversed. However, by induction, Z_{k-1} is symmetric, so we have $\overleftarrow{Z_{k-1}} = Z_{k-1}$. This establishes the statement for k.

Given (1), Item 1 follows immediately by induction.

For Item 2, we proceed by induction on k. As a base case, when $k = 1$, the statement follows by inspection. Now suppose that $k > 2$ and that the statement holds for $k - 1$.

Case 1: $i < 2^{k-1}$**.** First suppose that $i < 2^{k-1}$. Then for any $z < k - 1$,

$$\mathcal{N}_k(z, i) = \mathcal{N}_{k-1}(z, i) = \left\lfloor \frac{i + 2^z}{2^{z+1}} \right\rfloor$$

by induction, establishing the statement. Further, if $z = k - 1$ but $i < 2^{k-1}$, we have

$$\mathcal{N}_k(k-1, i) = 0 = \left\lfloor \frac{i + 2^{k-1}}{2^k} \right\rfloor,$$

and the statement again follows.

Case 2: $i \geq 2^{k-1}$**.** Next, we turn our attention to the case that $i \geq 2^{k-1}$. In this case, by (1), we have

$$\mathcal{N}_k(z, i) = \mathcal{N}_{k-1}(z, 2^{k-1} - 1) + \mathbf{1}[z = (k-1)] + \mathcal{N}_{k-1}(z, i - 2^{k-1}).$$

If $z < k-1$, then by induction we have

$$\mathcal{N}_k(z,i) = \left\lfloor \frac{2^{k-1} - 1 + 2^z}{2^{z+1}} \right\rfloor + \left\lfloor \frac{i - 2^{k-1} + 2^z}{2^{z+1}} \right\rfloor.$$

Suppose that $i = 2^{k-1} + \Delta_1 \cdot 2^{z+1} + \Delta_2$, where $\Delta_2 < 2^{z+1}$. Then we can write the above as:

$$\begin{aligned}\mathcal{N}_k(z,i) &= \left\lfloor 2^{k-z-2} + \frac{1}{2} - \frac{1}{2^{z+1}} \right\rfloor + \left\lfloor \Delta_1 + \frac{1}{2} + \frac{\Delta_2}{2^{z+1}} \right\rfloor \\ &= 2^{k-z-2} + \Delta_1 + \left\lfloor \frac{1}{2} + \frac{\Delta_2}{2^{z+1}} \right\rfloor,\end{aligned}$$

where above we have used the fact that $z < k-1$ and so 2^{k-z-2} is an integer. On the other hand, we have

$$\left\lfloor \frac{i + 2^z}{2^{z+1}} \right\rfloor = \left\lfloor 2^{k-z-2} + \Delta_1 + \frac{\Delta_2}{2^{z+1}} + \frac{1}{2} \right\rfloor = 2^{k-z-2} + \Delta_1 + \left\lfloor \frac{1}{2} + \frac{\Delta_2}{2^{z+1}} \right\rfloor,$$

which is the same. Thus, we conclude that if $z < k-1$,

$$\mathcal{N}_k(z,i) = \left\lfloor \frac{i + 2^z}{2^{z+1}} \right\rfloor,$$

as desired. On the other hand, if $z = k-1$, then by (1) we have $\mathcal{N}_k(k-1,i) = 1$ for all $i \geq 2^{k-1}$, and indeed this is equal to $\left\lfloor \frac{i+2^{k-1}}{2^k} \right\rfloor$. This completes the proof of Item 2. □

Definition 3 (Unary code)**.** The **Unary code** $\mathcal{U} \subseteq \mathbb{F}_2^\ell$ is defined as the image of the encoding map $\mathrm{Enc}_\mathcal{U} : \{0,\dots,\ell\} \to \mathbb{F}_2^\ell$ given by $\mathrm{Enc}_\mathcal{U}(j) := 1^j \circ 0^{\ell-j}$. The decoding map $\mathrm{Dec}_\mathcal{U} : \mathbb{F}_2^\ell \to \{0,\dots,\ell\}$ is given by

$$\mathrm{Dec}_\mathcal{U}(x) = \mathrm{argmin}_{j \in \{0,\dots,\ell\}} \Delta(x, \mathrm{Enc}_\mathcal{U}(j)).$$

Similarly, we define the **complementary Unary code** $\mathcal{U}^{\mathrm{comp}} \subseteq \mathbb{F}_2^\ell$ as the image of the encoding map $\mathrm{Enc}_{\mathcal{U}^{\mathrm{comp}}} : \{0,\dots,\ell\} \to \mathbb{F}_2^\ell$ given by $\mathrm{Enc}_{\mathcal{U}^{\mathrm{comp}}}(j) := 0^j \circ 1^{\ell-j}$. The decoding map $\mathrm{Dec}_{\mathcal{U}^{\mathrm{comp}}} : \mathbb{F}_2^\ell \to \{0,\dots,\ell\}$ is given by

$$\mathrm{Dec}_{\mathcal{U}^{\mathrm{comp}}}(x) = \mathrm{argmin}_{j \in \{0,\dots,\ell\}} \Delta(x, \mathrm{Enc}_{\mathcal{U}^{\mathrm{comp}}}(j)).$$

Naively, the runtime complexity of $\mathrm{Dec}_\mathcal{U}$ is $O(\ell^2)$, as one would loop over ℓ values of j and compute $\Delta(x, \mathrm{Enc}_\mathcal{U}(j))$ for each. However, this decoder can be implemented in time linear in ℓ, which is our next lemma.

Lemma 3. Let $\mathcal{U}$ be the unary code of length ℓ. Then $\mathrm{Dec}_\mathcal{U}$ and $\mathrm{Dec}_{\mathcal{U}^{\mathrm{comp}}}$ can be implemented to run in time $O(\ell)$.

Proof. We prove the statement just for $\mathrm{Dec}_\mathcal{U}$; it is the same for $\mathrm{Dec}_{\mathcal{U}^{\mathrm{comp}}}$. For a fixed j, by definition we have $\mathrm{Enc}_\mathcal{U}(j) = 1^j 0^{\ell-j}$. To compute $\Delta(x, \mathrm{Enc}_\mathcal{U}(j))$ for each j, one needs to count the number of zeros before index j and the number of ones after index j. We can express this as follows:

$$\Delta(x, \mathrm{Enc}_\mathcal{U}(j)) = \sum_{m=1}^{\ell} \mathbb{1}[x[m] = \mathrm{Enc}_\mathcal{U}(j)] = \sum_{m=0}^{j} \mathbb{1}[x[m] = 0] + \sum_{m=j+1}^{\ell} \mathbb{1}[x[m] = 1] \qquad (2)$$

Define the array $T[m]$ to count the number of zeros up to index m:

$$T[m] = \begin{cases} \mathbb{1}[x[m] = 0] & m = 1 \\ T[m-1] + \mathbb{1}[x[m] = 0] & m > 1 \end{cases}$$

This array can be computed in time $O(\ell)$. Using $T[m]$, we can rewrite $\Delta(x, \mathrm{Enc}_{\mathcal{U}}(j))$ as:

$$\Delta(x, \mathrm{Enc}_{\mathcal{U}}(j)) = T[j] + (\ell - j - (T[\ell] - T[j]))) \tag{3}$$

Thus, given the array $T[m]$, the distance for each j can be computed in $O(1)$ time. Therefore, the overall time complexity of $\mathrm{Dec}_{\mathcal{U}}$ is $O(\ell)$.

□

Next, we define the failure probability of a binary code.

Definition 4. Fix $p \in (0, 1)$. Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a code with message length k and encoding and decoding maps $\mathrm{Dec}_{\mathcal{C}}$ and $\mathrm{Enc}_{\mathcal{C}}$ respectively. The probability of failure of $\mathcal{C}$ is

$$P_{\text{fail}}(\mathcal{C}) = \max_{v \in \mathbb{F}_2^k} \Pr_{\eta_p}[\mathrm{Dec}_{\mathcal{C}}(\mathrm{Enc}_{\mathcal{C}}(v) + \eta_p) \neq v],$$

where the probability is over a noise vector $\eta_p \in \mathbb{F}_2^n$ with $\eta_p \sim \mathrm{Ber}(p)^n$.

Note that this definition is simply the block error probability of the binary code $\mathcal{C}$ one the binary symmetric channel with parameter p.

2.2 Base code

Ingredients. We begin by fixing an outer code and an inner code. Let $q = 2^{k'}$ for some integer k'. Let $\mathcal{C}_{\text{out}}$ be an $[n, k]_q$ linear code over $\mathbb{F}_q$. Denote the rate of $\mathcal{C}_{\text{out}}$ by $\mathcal{R}_{\text{out}} \in (0, 1)$ and the relative distance of $\mathcal{C}_{\text{out}}$ by $\delta_{\text{out}} \in (0, 1)$. Note that it is possible to decode $\mathcal{C}_{\text{out}}$ from e errors and t erasures as long as $2e + t < \delta_{\text{out}} n$. Let $\mathrm{Dec}_{\mathcal{C}_{\text{out}}} : (\mathbb{F}_q \cup \perp)^n \to \mathbb{F}_q^k$ denote the decoding map for $\mathcal{C}_{\text{out}}$ that can do this, where $\perp$ represents an erasure. (Later, we will choose $\mathcal{C}_{\text{out}}$ to be a Reed–Solomon code, so in particular $\mathrm{Dec}_{\mathcal{C}_{\text{out}}}$ can be implemented efficiently).

Let $\mathcal{C}_{\text{in}} \subseteq \mathbb{F}_2^{n'}$ be a linear code of dimension k'. We will abuse notation and use $\mathcal{C}_{\text{in}} : \{0,1\}^{k'} \to \{0,1\}^{n'}$ to also denote its encoding map. Let $\mathcal{R}_{\text{in}} = \frac{k'}{n'}$ denote the rate of $\mathcal{C}_{\text{in}}$. Let $\mathcal{C} = \mathcal{C}_{\text{out}} \circ \mathcal{C}_{\text{in}}$ denote the concatenation of $\mathcal{C}_{\text{out}}$ and $\mathcal{C}_{\text{in}}$, so that

$$\mathcal{C} = \{(\mathcal{C}_{\text{in}}(\sigma[1]), \ldots, \mathcal{C}_{\text{in}}(\sigma[n])) : \sigma \in \mathcal{C}_{\text{out}}\} \subseteq \mathbb{F}_2^{n \cdot n'},$$

where above we identify $\mathbb{F}_2^{k'}$ with $\mathbb{F}_q = \mathbb{F}_{2^{k'}}$ in the natural ($\mathbb{F}_2$-linear) way. Let $A \in \mathbb{F}_2^{k' \cdot k \times n' \cdot n}$ be the generator matrix of $\mathcal{C}$. Note that A can be obtained efficiently from the generator matrices of $\mathcal{C}_{\text{in}}$ and $\mathcal{C}_{\text{out}}$.

Throughout the paper, we shall use σ to denote an outer codeword and c to denote a codeword in the concatenated code $\mathcal{C}$. To ease notation, for $c \in \mathcal{C}$, we will denote $c[m] := \mathcal{C}_{\text{in}}(\sigma[m]) = c[(m-1) \cdot n' + 1 : m \cdot n'] \in \mathbb{F}_2^{n'}$. Namely, $c[m]$ is the mth inner codeword inside the concatenated codeword c. Similarly, for a row a of the generator matrix A, we will let $a[m] := a[(m-1) \cdot n' + 1 : m \cdot n']$. (Note that for any other string in the paper, when we write $x[m]$, we mean the mth *bit* in the string x; we use this notation only for codewords c in the concatenated code $\mathcal{C} = \mathcal{C}_{\text{out}} \circ \mathcal{C}_{\text{in}}$, including the rows of A).

Ordering the base code. We define an order on the codewords $c_0, c_1, \ldots, c_{2^{kk'}-1}$ of our concatenated code $\mathcal{C}$. Define c_0 to be the zero codeword. For $i > 0$, The ith codeword in $\mathcal{C}$ is defined by

$$c_i = A^T \mathcal{R}_{k'k}(i). \tag{4}$$

As $\mathcal{R}_{k'k}$ is a binary reflected code, $\mathcal{R}_{k'k}(i-1)$ and $\mathcal{R}_{k'k}(i)$ differ in exactly one index. Let z_i denote this index, so we have

$$\mathcal{R}_{k'k}(i-1)[z_i] \neq \mathcal{R}_{k'k}(i)[z_i] \ .$$

Denote by a_m the mth row of A. Then, for every $i \in \{1, 2, \ldots, 2^{kk'} - 1\}$, we have

$$c_i = c_{i-1} \oplus a_{z_i}. \tag{5}$$

This is clearly an ordering of all the codewords of $\mathcal{C}$. Indeed, $\mathcal{R}_{k'k}$ is a bijection and A is full-rank, so as i varies in $\{0, \ldots, 2^{kk'} - 1\}, c_i = A^T \mathcal{R}_{k'k}(i)$ varies over all the codewords in $\mathcal{C}$, hitting each $c \in \mathcal{C}$ exactly once.

Note that the ordering of $\mathcal{C}$ immediately implies an ordering of $\mathcal{C}_{\text{out}}$. Indeed, by the concatenation process, there is a bijection between $\mathcal{C}$ and $\mathcal{C}_{\text{out}}$. Thus, the ith codeword c_i in our concatenated codeword defines also the ith codeword in the outer code. We let $\sigma_i \in \mathcal{C}_{\text{out}}$ denote this outer codeword. That is,

$$c_i = \sigma_i \circ \mathcal{C}_{\text{in}} = (\mathcal{C}_{\text{in}}(\sigma_i[1]), \ldots, \mathcal{C}_{\text{in}}(\sigma_i[n])).$$

2.3 Intermediate Code

Next, we explain how to get our intermediate code $\mathcal{W}$ from our base codes $\mathcal{C}_{\text{out}}, \mathcal{C}_{\text{in}}$, and $\mathcal{C} = \mathcal{C}_{\text{out}} \circ \mathcal{C}_{\text{in}}$.

Encoding the generator matrix row difference. Recall that the difference of every two consecutive codewords is a row of the generator matrix A, namely, $c_i - c_{i-1} = a_{z_i}$. In the ith codeword of the intermediate code $\mathcal{W}$, we will include z_i, encoded with a repetition code that repeats each bit of $L/\log(kk')$ times. We shall explicitly state the value of L when we prove Theorem 1 and choose the parameters of our scheme. Since z_i can be represented using $\log(kk')$ bits, we shall encode z_i using the map[6] $\mathcal{L} : \mathbb{F}_2^{\log(kk')} \to \mathbb{F}_2^L$ which simply performs repetition encoding described above, to obtain

$$L_{z_i} = \mathcal{L}(z_i) \ .$$

Construction of $\mathcal{W}$. Now, we describe how to generate our intermediate code $\mathcal{W}$. Informally, to get the ith codeword $w_i \in \mathcal{W}$, we take the ith codeword $c_i \in \mathcal{C}$; add L_{z_i} at the beginning; and then break up c by including short strings of repeated bits in between each inner codeword $c_i[m] \in \mathcal{C}_{\text{in}}$. Formally, we have the following definition.

Definition 5. Let B be an integer that will be chosen later. Let $d := n'n + B(n+1) + L$. The intermediate code $\mathcal{W}$, along with its ordering, is defined as follows. For each $i \in \{0, \ldots, q^k - 1\}$, define $w_i \in \{0,1\}^d$ by the equation

$$w_i = \begin{cases} L_{z_i} \circ 0^B \circ c_i[1] \circ 0^B \circ \cdots \circ 0^B \circ c_i[n] \circ 0^B & \text{if } i \text{ is even} \\ L_{z_i} \circ 1^B \circ c_i[1] \circ 1^B \circ \cdots \circ 1^B \circ c_i[n] \circ 1^B & \text{if } i \text{ is odd} \end{cases} \tag{6}$$

[6]Note that $\log(kk')$ might not be an integer. Going forward, we will drop floors and ceilings in order to ease notation and the analysis. We note that the loss in the rate due to these roundings is negligible and does not affect the asymptotic results.

where c_i is the ith codeword in $\mathcal{C}$, and where we recall that $c_i[m] = \mathcal{C}_{\text{in}}(\sigma_i[m])$ denotes the mth inner codeword in c_i. Finally, we define $\mathcal{W} \subseteq \mathbb{F}_2^d$ by

$$\mathcal{W} = \{w_i \,:\, i \in \{0, 1, \dots, q^k - 1\}\}.$$

Note that $\mathcal{W}$ has the natural ordering $w_0, w_1, \dots, w_{q^k-1}$.

2.4 The Final Code

To create our robust Gray code $\mathcal{G}$, given any two consecutive codewords in $\mathcal{W}$, we inject extra codewords between them to create $\mathcal{G}$. Before we formally define this, we begin with some notation.

Definition 6 (The parameters $r_i, h_{i,j}, \bar{j}$)**.** Let $\mathcal{W} \subseteq \{0,1\}^d$ be a code defined as in Definition 5. For each $i \in \{0, \dots, q^k - 1\}$, define $r_i = \sum_{\ell=1}^{i} \Delta(w_{\ell-1}, w_\ell)$, and let $N = r_{q^k-1}$. Also, for $i \in \{0, \dots, q^k - 1\}$ and $1 \le j < \Delta(w_i, w_{i+1})$, let $h_{i,j} \in [d]$ be the jth index where codewords w_i and w_{i+1} differ. We will also define $h_i = (h_{i,1}, h_{i,2}, \dots, h_{i,\Delta(w_i,w_{i+1})-1}) \in [d]^{\Delta(w_i,w_{i+1})-1}$ to be the vector of all indices in which w_i and w_{i+1} differ, in order, except for the last one.[7] Finally, for $i \in \{0, \dots, q^k - 1\}$ and for $j \in [r_i, r_{i+1})$, we will use the notation $\bar{j}$ to denote $j - r_i$. That is, $\bar{j}$ is the index of j in the block $[r_i, r_{i+1})$ in which j falls.

With this notation, we are ready to define our robust Gray code $\mathcal{G}$.

Definition 7 (Definition of $\mathcal{G}$)**.** Define the zero'th codeword of $\mathcal{G}$ as $g_0 = w_0$. Fix $j \in \{1, \dots, N-1\}$. If $j = r_i$ for some i, we define $g_j \in \{0,1\}^d$ by $g_j = w_i$. On the other hand, if $j \in (r_i, r_{i+1})$ for some i, then we define $g_j \in \{0,1\}^d$ as

$$g_j = \text{pref}_{h_{i,\bar{j}}}(w_{i+1}) \circ \text{suff}_{h_{i,\bar{j}}+1}(w_i). \tag{7}$$

Finally, define $\mathcal{G} \subseteq \{0,1\}^d$ by $\mathcal{G} = \{g_j \,:\, j \in \{0, \dots, N-1\}\}$, along with the encoding map $\text{Enc}_{\mathcal{G}} : \{0, \dots, N-1\} \to \{0,1\}^d$ given by $\text{Enc}_{\mathcal{G}}(j) = g_j$.

Note that when $j \in [r_i, r_{i+1})$, the last bit that has been flipped to arrive at g_j in the ordering of $\mathcal{G}$ (that is, the "crossover point" alluded to in the introduction) is $h_{i,\bar{j}}$. We make a few useful observations about Definition 7. The first observation follows immediately from the definition.

Observation 3 ($\mathcal{G}$ is a Gray code)**.** $\mathcal{G}$ is a Gray code. That is, for any $j \in \{0, \dots, N-1\}$, we have that $\Delta(g_j, g_{j+1}) = 1$.

Next, we bound the rate of $\mathcal{G}$.

Observation 4 (Rate of $\mathcal{G}$)**.** The rate of the robust Gray code $\mathcal{G}$ defined in Definition 7 is at least

$$\frac{\mathcal{R}_{\text{out}}\mathcal{R}_{\text{in}}}{1 + \frac{B}{n'} \cdot (1 + \frac{1}{n}) + \frac{L}{nn'}}. \tag{8}$$

[7] The reason we don't include the last one is because of Definition 7 below, in which we flip bits one at a time to move between the codewords g_j of our robust Gray code $\mathcal{G}$. In more detail, the reason is because once the last differing bit has been flipped, g_j will lie in $[w_{i+1}, w_{i+2})$, not $[w_i, w_{i+1})$.

Proof. Recall that $\mathcal{C}_{\text{in}}$ has rate $\mathcal{R}_{\text{in}}$ and $\mathcal{C}_{\text{out}}$ has rate $\mathcal{R}_{\text{out}}$. Then the code $\mathcal{W}$ constructed as in Definition 5 has rate

$$\frac{\log q^k}{n' \cdot n + B(n+1) + L} = \frac{\mathcal{R}_{\text{out}} n \cdot \mathcal{R}_{\text{in}} n'}{n' \cdot n + B(n+1) + L}$$
$$= \frac{\mathcal{R}_{\text{out}} \mathcal{R}_{\text{in}}}{1 + \frac{B}{n'} \cdot (1 + \frac{1}{n}) + \frac{L}{nn'}} \ .$$

Thus, the rate of $\mathcal{G}$ is at least the above, given that $\mathcal{G}$ has more codewords than $\mathcal{W}$ but the same length. □

Fix i, and suppose that g_j is obtained as an intermediate codeword between w_i and w_{i+1}. Then, on the coordinates in which w_i and w_{i+1} differ, g_j will disagree with w_i for a first chunk of them, and agree with w_i for the rest. We make this precise in the following observation.

Observation 5. Let $g_j \in \mathcal{G}$, and suppose that $j \in (r_i, r_{i+1})$ for some $i \in \{0, \dots, q^k - 1\}$. Recall from Definition 6 that $h_i \in [d]^{\Delta(w_i, w_{i+1})-1}$ is the vector of positions on which w_i and w_{i+1} differ (except the last one). Then

$$(g_j + w_i)[h_i] = \text{Enc}_{\mathcal{U}}(\bar{j}),$$

where $\mathcal{U} \subset \{0,1\}^{\Delta(w_i, w_{i+1})-1}$ is the unary code of length $\Delta(w_i, w_{i+1}) - 1$. Above, $(g_j + w_i)[h_i]$ denotes the restriction of the vector $g_i + w_i \in \mathbb{F}_2^d$ to the indices that appear in the vector h_i.

Further, for every $m \geq \bar{j}$, we have

$$(g_j + w_i)[h_i[1 : m]] = \text{Enc}_{\mathcal{U}}(\bar{j}),$$

where $\mathcal{U}$ is the unary code of length m. That is, even if we take the first m values of h_i, then as long as $m \geq \bar{j}$, the restriction of $(g_j + w_i)$ to these values match the unary encoding of $\bar{j}$.

Proof. By definition, h_i contains the indices on which w_i and w_{i+1} differ, and also by definition, by the time we have reached g_j, the first $j - r_i = \bar{j}$ of these indices have been flipped from agreeing with w_i to agreeing with w_{i+1}. Thus, if we add g_j and w_i (mod 2), we will get 1 on the first $j - r_i$ indices and 0 on the on the rest. The "further" part follows immediately. □

Our next objective is to show that Definition 7 actually defines an injective map. We begin by providing some notation for different parts of the codeword $g_j \in \mathcal{G}$. For a string x, $x[m : m']$ denotes the substring $(x_m, x_{m+1}, \dots, x_{m'})$. For any $x \in \{0,1\}^d$ define

- $\tilde{L} = x[1 : L]$,
- $s_m = x[L + (m-1)(B + n') + 1 : L + (m-1)(B + n') + B]$ for $m \in [n+1]$,
- $\tilde{c}_m = x[L + mB + (m-1)n' + 1 : L + m(B + n')]$ for $m \in [n]$,

As a result, any string x that is either a codeword or a corrupted codeword, has the following format (see also Figure 1):

$$x = \tilde{L} \circ s_1 \circ \tilde{c}_1 \circ \dots \tilde{c}_n \circ s_{n+1} \ . \tag{9}$$

We will call each of $\tilde{L}_i, s_i, c_i$ a **chunk**. For a codeword $g_j \in \mathcal{G}$ such that $j \in [r_i, r_{i+1})$ we will call a chunk a **full chunk** if it is equal to its corresponding chunk in either w_i or w_{i+1}. This notation is illustrated in Figure 1.

The following lemma shows that for each g_j there is at most a single chunk that is not a full chunk.

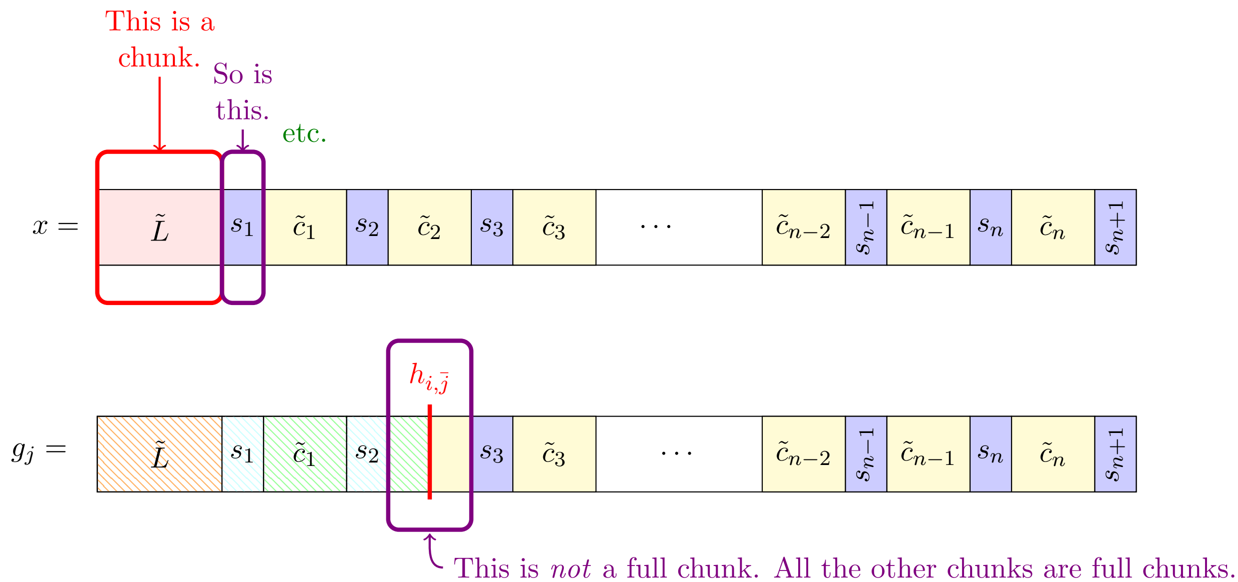


Figure 1: The notation used to break up vectors $x \in \{0,1\}^d$ into chunks (top), and the distinction between chunks and full chunks when x happens to be a codeword g_j (bottom). Notice that for g_j, if $j \in [r_i, r_{i+1})$ then we have, e.g., $s_m = b_i^B$ and $\tilde{c}_m = c_i[m]$ or $c_{i+1}[m]$, whenever the corresponding chunks are full chunks.

Lemma 4. Fix $j \in \{0, \ldots, N-1\}$. Suppose that $i \in \{0, \ldots, q^k - 1\}$ is such that $j \in [r_i, r_{i+1})$, so $g_j \in \mathcal{G}$ can be written as $g_j = \tilde{L} \circ s_1 \circ c_1 \circ \ldots c_n \circ s_{n+1}$ as above. Then at most one of the substrings in $\mathcal{S} = \{\tilde{L}, s_1 \ldots, s_{n+1}, c_1, \ldots, c_n\}$ is not equal to the corresponding substring in w_i or w_{i+1}.

Proof. First, suppose that $j = r_i$. Then in that case $g_j = w_i$ and all of the substrings in $\mathcal{S}$ are equal to their corresponding substring. Otherwise, $j \in (r_i, r_{i+1})$. In that case, $\bar{j} \in [1, r_{i+1} - r_i) = [1, \Delta(w_i, w_{i+1}))$. This means that $h_{i,\bar{j}}$ (the "crossover point" for g_j) is defined, and indexes a position in g_j, and in particular in one of the sub-strings in $\mathcal{S}$. Then other substrings strictly to the left of $h_{i,\bar{j}}$ are equal to their corresponding substring in w_{i+1}; and the ones strictly to the right are equal to the corresponding substring in w_i. □

Next, we show that there are no "collisions" in $\mathcal{G}$; that is, there are no $j \neq j'$ so that $g_j = g_{j'}$.

Lemma 5. Let $\mathcal{G}$ and $\mathrm{Enc}_{\mathcal{G}}$ be as in Definition 7. Then $\mathrm{Enc}_{\mathcal{G}}$ is injective.

Proof. Assume, for the sake of contradiction, that there are two distinct $j, j' \in \{0, \ldots, N-1\}$ such that $g_j = g_{j'}$. Without loss of generality assume that $j' > j$. There are three scenarios possible.

1. **Case 1:** Both j and j' are in the interval $[r_i, r_{i+1})$. Then we claim that $g_j[h_{i,\bar{j}'}] \neq g_{j'}[h_{i,\bar{j}'}]$. The reason is that $g_j[h_{i,\bar{j}'}] = w_i[h_{i,\bar{j}'}]$ and $g_{j'}[h_{i,\bar{j}'}] = w_{i+1}[h_{i,\bar{j}'}]$. This implies that $w_i[h_{i,\bar{j}'}] = w_{i+1}[h_{i,\bar{j}'}]$ which contradicts the definition of $h_{i,\bar{j}'}$.

2. **Case 2:** $j \in [r_{i-1}, r_i)$ and $j' \in [r_i, r_{i+1})$. Then g_j is an interpolation of w_{i-1} and w_i, and $g_{j'}$ is an interpolation of w_i and w_{i+1}. Denote

$$g_j = \tilde{L} \circ s_1 \circ \tilde{c}_1 \circ \ldots \tilde{c}_n \circ s_{n+1}$$

and

$$g_{j'} = \tilde{L}'_1 \circ s'_1 \circ \tilde{c'}_1 \circ \ldots \tilde{c'}_n \circ s'_{n+1} \; .$$

If $h_{i,\bar{j}}$ does not fall into $\tilde{L}$ or s_{n+1} then $(s_1, \ldots, s_n, s_{n+1})$ cannot be equal to $(s'_1, \ldots, s'_n, s'_{n+1})$. Indeed, assuming without loss of generality that i is even, then $(s_1, \ldots, s_{n+1}) = 0^a 1^b$ where both a and b are nonzero, while $(s'_1, \ldots, s'_n, s'_{n+1})$ is of the form $1^{a'} 0^{b'}$. An identical argument shows that if $h_{i+1,\bar{j}'}$ does not fall into f'_1 or s'_{n+1} then $(s_1, \ldots, s_n, s_{n+1})$ cannot be equal to $(s'_1, \ldots, s'_n, s'_{n+1})$. We are left with the case where $h_{i,\bar{j}}$ falls in $\tilde{L}$ or s_{n+1} and $h_{i+1,\bar{j}'}$ falls in f'_1 or s'_{n+1}. In this case, since the parities of $i-1$ and i are different, the only possibility to get equality between $(s_1, \ldots, s_{n+1})$ and $(s'_1, \ldots, s'_{n+1})$ is if $h_{i,\bar{j}}$ is in $\tilde{L}$ and $h_{i+1,\bar{j}'}$ falls exactly on the last bit of s'_{n+1}. This implies that $(\tilde{c}_1, \ldots, \tilde{c}_n)$—which corresponds to the outer codeword σ_{i-1}—and $(\tilde{c'}_1, \ldots, \tilde{c'}_n)$—which corresponds to the outer codeword σ_{i+1}—are equal, a contradiction of the fact that the codewords in our ordering of the outer code are all distinct.

3. **Case 3:** $j \in [r_i, r_{i+1})$ and $j' \in [r_{i'}, r_{i'+1})$ where $|i - i'| > 1$. As before, denote

$$g_j = \tilde{L} \circ s_1 \circ \tilde{c}_1 \circ \ldots \tilde{c}_n \circ s_{n+1}$$

and

$$g_{j'} = \tilde{L}'_1 \circ s'_1 \circ \tilde{c'}_1 \circ \ldots \tilde{c'}_n \circ s'_{n+1} \; .$$

By Lemma 4, only a single chunk in g_j (resp. $g_{j'}$) is not equal to the corresponding chuck in w_i or w_{i+1} (resp. $w_{i'}$ or $w_{i'+1}$). We shall consider several sub-cases depending on the locations of $h_{i,\bar{j}}$ and $h_{i',\bar{j}'}$.

First, assume that $h_{i,\bar{j}}$ falls into $s_m \circ \tilde{c}_m$ and $h_{i',\bar{j}'}$ into $s'_{m'} \circ \tilde{c'}_{m'}$ where $m \neq m' \in [n]$. Also, assume without loss of generality that $m' > m$. Note that since neither $h_{i,\bar{j}}$ nor $h_{i',\bar{j}'}$ fall in the last chunks (s_{n+1} and s'_{n+1}, respectively), it must be that i and i' have the same parity; otherwise the chunks s_{n+1} and s'_{n+1} would disagree, contradicting our assumption that $g_j = g_{j'}$. Assume that $(s_1, \ldots, s_{n+1})$ and $(s'_1, \ldots, s'_{n+1})$ are of the form $1^a 0^b$ and $1^{a'} 0^{b'}$, respectively. Clearly, as $h_{i,\bar{j}}$ falls into $s_m \circ \tilde{c}_m$ and $h_{i',\bar{j}'}$ into $s'_{m'} \circ \tilde{c'}_{m'}$, and $m' > m$, it must be that $a' > a$. We conclude that $g_j \neq g_{j'}$, a contradiction.

Now assume that both $h_{i,\bar{j}}$ and $h_{i',\bar{j}'}$ fall into $s_m \circ \tilde{c}_m$ and $s'_m \circ \tilde{c'}_m$, respectively. (Note that the difference between this sub-case and the previous one is that $h_{i,\bar{j}}$ and $h_{i',\bar{j}'}$ fall into chunks with the *same* index m). In this case, since $\tilde{L}$ and f'_1 in g_j and $g_{j'}$ are full chunks, it holds that the tuple

$$(L_{z_{i+1}}, c_{i+1}[1], \ldots, c_{i+1}[m-1], c_i[m+1], \ldots, c_i[n])$$

is equal to

$$(L_{z_{i'+1}}, c_{i'+1}[1], \ldots, c_{i'+1}[m-1], c_{i'}[m+1], \ldots, c_{i'}[n]) \; .$$

Now, since $L_{z_{i+1}} = L_{z_{i'+1}}$, we have that c_{i+1} and $c_{i'+1}$ are obtained by adding the same row a_z of the generator matrix A, to c_i and $c_{i'}$, respectively. Thus, for each $r \leq m-1$ we have that $c_i[r] = c_{i'}[r]$ and in total,

$$(c_i[1], \ldots, c_i[m-1], c_i[m+1], \ldots, c_i[n]) = (c_{i'}[1], \ldots, c_{i'}[m-1], c_{i'}[m+1], \ldots, c_{i'}[n])$$

which contradicts the fact that $i \neq i'$ and that the minimum distance of $\mathcal{C}_{\text{out}}$ satisfies $\delta_{\text{out}} n > 1$. Finally, we consider the sub0case where $h_{i,\bar{j}}$ falls in $\tilde{L}$ or s_{n+1}. In this case, if $h_{i',\bar{j}'}$ also falls in f'_1 or s'_{n+1}, then $(\tilde{c}_1, \ldots, \tilde{c}_n)$ and $(\tilde{c'}_1, \ldots, \tilde{c'}_n)$ correspond to two distinct outer codewords, which implies that $g_j \neq g_{j'}$, contradicting our assumption that they are the same. If $h_{i',\bar{j}'}$ doesn't fall in f'_1 or s'_{n+1}, then it must fall into an s'_m or $\tilde{c'}_m$ for some $m \in [n]$. In this case, $(s_1, \ldots, s_n, s_{n+1})$ will be the all 1 or all 0 string but $(s'_1, \ldots, s'_n, s'_{n+1})$ clearly cannot be the all 0 or 1 string since $s'_{n+1} \neq s'_m$.

Thus, in all cases we arrive at a contradiction, and this completes the proof. □

3 Decoding Algorithm

In this section, we define the decoding algorithm. In the following paragraphs, we will give a high level overview of the major steps in the decoding procedures. We denote the input to the algorithm by $x \in \mathbb{F}_2^d$, and we recall that x is of the following form (see also Figure 1):

$$x = \tilde{L} \circ s_1 \circ \tilde{c}_1 \circ \ldots \circ \tilde{c}_n \circ s_{n+1} .$$

Recall that x is a noisy version of some codeword of $\mathcal{G}$; let us write $x = g_j \oplus \eta$ for a noise vector $\eta \in \mathbb{F}_2^d$, so our objective is to return $\hat{j} \approx j$. As usual, suppose that $j \in [r_i, r_{i+1})$, and define $\bar{j} = j - r_i$, so that $h_{i,\bar{j}}$ is the crossover point in the correct codeword g_j.

Our primary decoding algorithm, $\text{Dec}_{\mathcal{G}}$, is given in Algorithm 1. The first objective of the decoding algorithm is to estimate the chunk in which the crossover point $h_{i,\bar{j}}$ occurs. We define $\ell \in \{0, \ldots, n+1\}$ to be

$$\ell = \begin{cases} 0 & \text{if } h_{i,\bar{j}} \text{ falls in } \tilde{L} \\ m & \text{if } h_{i,\bar{j}} \text{ falls in } s_m \circ \tilde{c}_m \text{ for } m \in [n] \\ n+1 & \text{if } h_{i,\bar{j}} \text{ falls in } s_{n+1} \end{cases} . \tag{10}$$

Intuitively speaking, ℓ will be the crossover point at the level of chunks. Algorithm 1 will estimate ℓ, and we will denote this estimation by $\hat{\ell}$.

Next, Algorithm 1 decodes each chunk $\tilde{c}_m$ using the inner code's decoding algorithm to obtain an estimate $\hat{\sigma}[m] \in \mathbb{F}_q$. Then, based on the location of $\hat{\ell}$ and the decoded symbols $\hat{\sigma}[m]$, we either invoke Algorithm 2 (`get-estimate`), or Algorithm 3 (`get-estimate-boundary`) in order to obtain our final estimate $\hat{j}$.

In more detail, for an appropriate constant $\beta \in (0, 1)$, we will show that with high probability, ℓ cannot be more than βn "far" from $\hat{\ell}$. We break up both our algorithm and analysis into two cases, depending on whether $\hat{\ell}$ lands in $(\beta n, n - \beta n)$. If $\hat{\ell} \in (\beta n, n - \beta n)$, we say that $\hat{\ell}$ is *in the middle*. In this case, we call Algorithm 2 to recover $\hat{j}$. If $\hat{\ell} \notin (\beta n, n - \beta n)$, we say that $\hat{\ell}$ is *in the boundaries*. In this case, we call Algorithm 3 to recover $\hat{j}$. We next describe Algorithm 2 and Algorithm 3, and why we break things into these two cases.

Algorithm 2 (`get-estimate`) is called when $\hat{\ell} \in (\beta n, n - \beta n)$. The first thing it does is to update our estimate $\hat{\sigma}$—which corresponds to an interpolation between two codewords of $\mathcal{C}_{\text{out}}$—to obtain a version $\hat{\sigma}$ that corresponds to only one codeword in $\mathcal{C}_{\text{out}}$. To do this, it first decodes the first L

bits to get z_{i+1} and uses this to update $\hat{\sigma}$ by:

$$\hat{\sigma}[m] = \begin{cases} \hat{\sigma}[m] - a_{z_{i+1}}[m] & \text{if } m < \hat{\ell}_s \\ \perp & \text{if } m \in [\hat{\ell}_s, \hat{\ell}_e] \\ \hat{\sigma}[m] & \text{if } m > \hat{\ell}_e \end{cases}.$$

where we use the $\perp$ symbol to indicate an erasure. Above, $\hat{\ell}_s = \hat{\ell} - \beta n$, $\hat{\ell}_e = \hat{\ell} + \beta n$ and recall that $a_{z_{i+1}}[m] = a_{z_{i+1}}[(m-1)n' + 1 : mn']$. Also, above we have used the fact that $a_{z_{i+1}}[m] \in \mathcal{C}_{\text{in}}$, and thus corresponds to some element of $\mathbb{F}_q$, so we treat $a_{z_{i+1}}[m]$ as an element of $\mathbb{F}_q$ in the subtraction above. Intuitively, what the algorithm is doing here is translating the elements of $\hat{\sigma}$ that correspond to c_{i+1} into elements that correspond to c_i. Finally, Algorithm 2 uses $\mathcal{C}_{\text{out}}$'s decoder on $\hat{\sigma} \in \mathbb{F}_q^n$ to obtain $\hat{i}$. Given $\hat{i}$, it computes $\hat{j}$ by taking into consideration how many bits were flipped from $w_{\hat{i}}[H]$ to get $x[H]$, where $H = \{i \mid w_{\hat{i}} \neq w_{\hat{i}+1}\}$.

Algorithm 3 (`get-estimate-boundary`) is invoked when $\hat{\ell} \notin (\beta n, n - \beta n)$. The general strategy in this algorithm is similar to that of Algorithm 2, but there are several differences. The main reason for these differences is that if $\hat{\ell}$ is in the boundaries, $\hat{\ell}$ will only be "close" to ℓ modulo n. To see intuitively why this should be true, consider two scenarios, one where j is all the way at the *end* of the interval $[r_i, r_{i+1})$, and a second where j is all the way at the *beginning* of the next interval $[r_{i+1}, r_{i+2})$. The j's in these two scenarios are close to each other, and their corresponding encodings under $\mathcal{G}$ are also close in Hamming distance. However, in the first scenario, ℓ is close to $n+1$, while in the second scenario, ℓ is close to 0. Thus, we should only expect to be able to estimate ℓ modulo n, and it could be possible that, for example, $\hat{\ell}$ is close to zero while ℓ is close to n.

Here is how we take this into account in Algorithm 3, relative to Algorithm 2 discussed above. First, we define $\hat{\ell}_s$ and $\hat{\ell}_e$ slightly differently, taking them modulo n as per the intuition above (see Figure 2). Second, Algorithm 3 sets $\hat{\sigma}_i[m]$ differently. For $m \in [1, \hat{\ell}_s] \cup [\hat{\ell}_e, n]$ we set $\hat{\sigma}_i[m] = \perp$. A crucial observation is that for every $m \in [\hat{\ell}_e, \hat{\ell}_s]$, if $\hat{\ell} \leq \beta n$, then $\tilde{c}_m$ is a corrupted version of $c_i[m]$ and if $\hat{\ell} \geq n - \beta n$ then $\tilde{c}_m$ is a corrupted version of $c_{i+1}[m]$. Since we could have *either* $\ell \leq \beta n$ *or* $\ell \geq n - \beta n$, we thus take *both* of these cases into account, and consider both c_i and c_{i+1} as possibilities. To this end, we compute *two* possible decodings of $\hat{\sigma}$, and we then get *two* options for $\hat{j}$, call them $\hat{j}_1$ and $\hat{j}_2$, by performing the same steps as in Algorithm 2 to each case. Then Algorithm 3 sets $\hat{j}$ to be the more likely of $\hat{j}_1$ and $\hat{j}_2$.

Finally, we discuss our last helper function, Algorithm 4, called `compute-r`. This helper function is called in both Algorithms 2 and 3. Its job is to compute r_i given i. While this seems like it should be straightforward—after all, r_i is defined in Definition 6 as a simple function of i—doing this efficiently without storing a lookup table of size q^k requires some subtlety. The key insight—and the reason that we defined the order on $\mathcal{C}$ the way we did—is that from (5), we have

$$c_i = c_{i-1} \oplus a_{z_i},$$

where we recall that z_i is the index in which $\mathcal{R}_k(i)$ and $\mathcal{R}_k(i-1)$ differ, and a_{z_i} is the z_i'th row of the generator matrix A of $\mathcal{C}$. To see why this matters, recall from Definition 6 that

$$r_i = \sum_{t=1}^{i} \Delta(w_{t-1}, w_t). \tag{11}$$

There are contributions to each $\Delta(w_{t-1}, w_t)$ from each of the chunks $\tilde{L}$, s_m, and $\tilde{c}_m$. Here, we discuss just the contribution from the $\tilde{c}_m$ chunks, as this illustrates the main idea. Due to (5), this contribution is

$$\sum_{t=1}^{i} \Delta(\sigma_{t-1} \circ \mathcal{C}_{\text{in}}, \sigma_t \circ \mathcal{C}_{\text{in}}) = \sum_{t=1}^{i} \|a_{z_t}\|. \tag{12}$$

We cannot afford to add up all of the terms in the sum individually, as i may be as large as q^k. However, instead we can compute the number of times that a particular row a_z appears in the sum above (this is given by Observation 2), and add $\|a_z\|$ that many times. As there are only $k \cdot k'$ such rows, this can be done efficiently.

This wraps up our informal description the decoding algorithm $\text{Dec}_{\mathcal{G}}$ and its helper functions; we refer the reader to the pseudocode for formal descriptions. In the next section, we present the analysis of $\text{Dec}_{\mathcal{G}}$.

Algorithm 1 $\text{Dec}_{\mathcal{G}}$: Decoding algorithm for $\mathcal{G}$

Input: $x \in \mathbb{F}_2^d$
    ▷ Estimate location of broken chunk:
for $m \in \{1, \dots, n+1\}$ **do**
    $\hat{s}_m = \text{Maj}(s_m)$
end for
$\hat{s} = (\hat{s}_1, \dots, \hat{s}_{n+1})$
$\hat{\ell}_1 = \text{Dec}_{\mathcal{U}}(\hat{s})$
$\hat{\ell}_2 = \text{Dec}_{\mathcal{U}^{\text{comp}}}(\hat{s})$
$\hat{\ell} = \begin{cases} \hat{\ell}_1 & \Delta(\hat{s}, 1^{\hat{\ell}_1} 0^{n+1-\hat{\ell}_1})) < \Delta(\hat{s}, 0^{\hat{\ell}_2} 1^{n+1-\hat{\ell}_2}) \\ \hat{\ell}_2 & \text{else} \end{cases}$
    ▷ Decode inner code $\mathcal{C}_{\text{in}}$:
for $m \in \{1, \dots, n\}$ **do**
    $\hat{\sigma}[m] = \text{Dec}_{\mathcal{C}_{\text{in}}}(\tilde{c}[m])$
end for
    ▷ Estimate j:
if $\hat{\ell} \in (\beta n, n - \beta n)$ **then**
    $\hat{j} =$ `get-estimate`$(x, \hat{\sigma}, \hat{\ell})$
else
    $\hat{j} =$ `get-estimate-boundary`$(x, \hat{\sigma}, \hat{\ell})$
end if
Output: $\hat{j}$

Algorithm 2 `get-estimate`: Computing the final estimate of $\hat{j}$

Require: $x \in \mathbb{F}_2^d, \hat{\sigma} \in \mathbb{F}_q^n, \hat{\ell} \in \{0, 1, \ldots, n+1\}$

▷ Calculate erasure interval:

$\hat{\ell}_s = \hat{\ell} - \beta n$

$\hat{\ell}_e = \hat{\ell} + \beta n$

▷ Update $\hat{\sigma}$, taking into account the estimate of the crossover point:

$\hat{z} = \mathrm{Dec}_{\mathcal{L}}(\tilde{L})$

for $m < \hat{\ell}_s$ **do**

  $\hat{\sigma}[m] = \hat{\sigma}[m] - a_{\hat{z}}[m]$

  ▷ $a_{\hat{z}}[m] \in \mathbb{F}_2^{n'}$ corresponds to an elt. of $\mathcal{C}_{\mathrm{in}}$ and hence of $\mathbb{F}_q$. Here, we treat $a_{\hat{z}}[m] \in \mathbb{F}_q$.

end for

for $m \geq \hat{\ell}_s$ and $m \leq \hat{\ell}_e$ **do**

  $\hat{\sigma}[m] = \perp$ ▷ Set $\hat{\sigma}[m]$ to an erasure if m is close to $\hat{\ell}$.

end for

for $m > \hat{\ell}_e$ **do**

  $\hat{\sigma}[m] = \hat{\sigma}[m]$ ▷ Don't update $\hat{\sigma}[m]$.

end for

▷ Decode outer code to obtain $\hat{i}$:

$\hat{i} = \mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}(\hat{\sigma})$

▷ Compute $\bar{j}$ and final estimate $\hat{j}$:

$H = \{m \mid w_{\hat{i}}[m] \neq w_{\hat{i}+1}[m]\}$

$\hat{\bar{j}} = \mathrm{Dec}_{\mathcal{U}}(x[H] \oplus w_{\hat{i}}[H])$

$\hat{j} = \texttt{compute-r}(\hat{i}) + \hat{\bar{j}}$

return $\hat{j}$

Algorithm 3 `get-estimate-boundary`: Computing the final estimate of $\hat{j}$ in the case where $\hat{\ell}$ lies in the boundary

Require: $x \in \mathbb{F}_2^d, \hat{\sigma} \in \mathbb{F}_q^n, \hat{\ell} \in \{0, 1, \ldots, n+1\}$

▷ Calculate erasure interval:

if $\hat{\ell} \leq \beta n$ **then**
  $\hat{\ell}_e = \hat{\ell} + \beta n$
  $\hat{\ell}_s = n + 1 + (\hat{\ell} - \beta n)$
else
  $\hat{\ell}_e = \hat{\ell} + \beta n - (n+1)$
  $\hat{\ell}_s = \hat{\ell} - \beta n$
end if

▷ Erase symbols too near the boundary:

for $m \in [0, \hat{\ell}_e] \cup [\hat{\ell}_s, n]$ **do**
  $\hat{\sigma}[m] = \perp$
end for

▷ Decode outer code to obtain $\hat{i}$:

$\hat{i} = \mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}(\hat{\sigma})$

▷ Case 1: ℓ is in the beginning:

$H = \{m \mid w_{\hat{i}}[m] \neq w_{\hat{i}+1}[m] \text{ and } m < L + 2\beta n(n' + B)\}$
$\hat{\hat{j}}_1 = \mathrm{Dec}_{\mathcal{U}}(x[H] \oplus w_{\hat{i}}[H])$
$\hat{j}_1 =$ `compute-r`$(\hat{i}) + \hat{\hat{j}}_1$

▷ Case 2: ℓ is towards the end:

$H = \{m \mid w_{\hat{i}}[m] \neq w_{\hat{i}-1}[m], m \geq d - 2\beta n(n' + B)\}$
$\hat{\hat{j}}_2 = \mathrm{Dec}_{\mathcal{U}^{\mathrm{comp}}}(x[H] \oplus w_{\hat{i}}[H])$
$\hat{j}_2 =$ `compute-r`$(\hat{i}) - \hat{\hat{j}}_2$

▷ Choose the most likely estimate:

$\hat{j} = \mathrm{argmin}_{\hat{j} \in \{\hat{j}_1, \hat{j}_2\}}(\Delta(x, \mathrm{Enc}_{\mathcal{G}}(\hat{j})))$
return $\hat{j}$

Algorithm 4 `compute-r`: Compute r_i, given i.

Input: $i \in \{0, \ldots, 2^{k'k} - 1\}$
$\hat{r}_i = i \cdot (n+1) \cdot B$
for $z \in \{0, \ldots, k'k - 1\}$ **do**
  $\hat{r}_i = \hat{r}_i + \lfloor \frac{i + 2^z}{2^{z+1}} \rfloor \left(\|a_z\| + \frac{2L}{\log(kk')} \cdot \|\mathrm{bin}(z)\| \right)$ ▷ a_z is the z'th row of A
  ▷ $\mathrm{bin}(z)$ is the binary expansion of z
  ▷ $\|\cdot\|$ denote Hamming weight
end for
Return: $\hat{r}_i$

4 Analysis

In this section we analyze Algorithm 1, proving a few statements that will be useful for our final proof of Theorem 1 in Section 5. We start by setting up a bit more notation. Throughout this section, we assume that the codeword that was transmitted was $g_j = \mathrm{Enc}_{\mathcal{G}}(j) \in \mathcal{G}$, for some integer $j \in [r_i, r_{i+1})$ for some i.

4.1 Running time of Algorithm 1

We begin by analyzing the running time of Algorithm 1. In particular, we prove the following proposition.

Proposition 1. For a code $\mathcal{D}$ of length D, let $T_{\mathrm{Enc}_{\mathcal{D}}}(D)$ and $T_{\mathrm{Dec}_{\mathcal{D}}}(D)$ denote the running time of $\mathcal{D}$'s encoding map $\mathrm{Enc}_{\mathcal{D}}$ and $\mathcal{D}$'s decoding map $\mathrm{Dec}_{\mathcal{D}}$, respectively. Given the codes $\mathcal{C}_{\mathrm{out}}, \mathcal{C}_{\mathrm{in}}$ and our Gray code $\mathcal{G}$ defined in Definition 7, it holds that:

1. $\mathrm{Enc}_{\mathcal{G}}$ runs in time
$$O\left(d^2\right) + O\left(T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}}(n) + n \cdot T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}}(n')\right)$$
2. $\mathrm{Dec}_{\mathcal{G}}$, which is given by Algorithm 1, runs in time
$$O(n \cdot B) + O\left(n \cdot T_{\mathrm{Dec}_{\mathcal{C}_{\mathrm{in}}}}(n')\right) + O\left(T_{\mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}}(n)\right) + O(d) \ .$$

Proof. We start with the encoding of $\mathcal{G}$, which consists of the following steps.

- Given an integer j, we need to compute the i for which $j \in [r_i, r_{i+1})$. Recall that given i, `compute-r` computes r_i. Thus, ind the corresponding i by performing binary search on the domain $i \in \{0, \dots, 2^{kk'} - 1\}$, calling `compute-r` in each iteration. Thus, the complexity of this step is $O(kk')$ times the time it takes to perform `compute-r`.

  We are left with analyzing the complexity of `compute-r`. The loop inside it runs for kk' iterations and in every iteration we perform a constant number of operations (multiplication, addition, and division) on kk'-bit integers. Note also that $||a_z||$ and $||\mathrm{bin}(z)||$ can be computed in $O(kk')$. Now, as multiplication of two kk'-bit integers can be done in $O(kk' \log(kk'))$ time [HVDH21], the total running time of `compute-r` is $O((kk')^2 \cdot \log(kk'))$.

  In total, the running time to find i given j is $O((kk')^3 \cdot \log(kk')) \leq \tilde{O}(d^3)$.

- Given i from the previous step, we encode i to c_i by first computing the message $\mathcal{R}_{kk'}(i) \in \mathbb{F}_2^{kk'}$. This can be done by simply invoking the recursive definition given in Definition 1 which runs in time $O(kk') = O(d)$. Then we encode the message $\mathcal{R}_{kk'}(i)$ with $\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}$ and $\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}$ to a codeword $c_i \in F_2^{nn'}$. This can be performed in time $O(T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}}(n) + n \cdot T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}}(n'))$. Thus, the final complexity of this step is $O(d) + O(T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}}(n) + n \cdot T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}}(n'))$.

- Given c_i from the previous step, we next compute $w_i \in \mathcal{W}$. This involves computing z_i and encoding it with the repetition code to obtain $L_{z_i} = \mathcal{L}(z_i)$; and computing the "buffer" sections s_m. Adding the buffers and encoding z_i clearly take time $O(d)$. Computing z_i from i can be done in time $O(d)$ as follows. We compute $\mathcal{R}_{kk'}(i-1)$ in time $O(d)$, and since we already computed $\mathcal{R}_{kk'}(i-1)$ in the previous step, we can identify z_i by searching the only bit that differs between $\mathcal{R}_{kk'}(i-1)$ and $\mathcal{R}_{kk'}(i)$.

- At the end of the previous step, we have w_i. We can repeat the process to obtain w_{i+1}. Then we may obtain g_j in time $O(d)$ from w_{i+1} by flipping $j - r_i$ bits of w_i (namely, the first $j - r_i$ bits on which w_i and w_{i+1} differ).

Thus the overall running time of the encoder $\mathrm{Enc}_{\mathcal{G}}$ is

$$\tilde{O}\left(d^3\right) + O\left(T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}}(n) + n \cdot T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}}(n')\right)$$

We proceed to analyze the running time of the decoder $\mathrm{Dec}_{\mathcal{G}}$, given in Algorithm 1. We go line-by-line through Algorithm 1.

1. In Line 3, we take the majority of B bits, for each $m \in [n+1]$. This takes time $O(nB)$.
2. In lines 6-8, we compute $\hat{\ell}$. This takes time $O(n)$, as we apply $\mathrm{Dec}_{\mathcal{U}}$ and $\mathrm{Dec}_{\mathcal{U}^{\mathrm{comp}}}$ once each to a vector of length $n+1$; and then compute two Hamming distances between vectors of length $n+1$. By Lemma 3, the former takes time $O(n)$, and the latter clearly also takes time $O(n)$.
3. In Lines 9-11, Algorithm 1 decodes n inner codewords. This takes time $O(n \cdot T_{\mathrm{Dec}_{\mathcal{C}_{\mathrm{in}}}}(n'))$.
4. In Lines 13-15, Algorithm 1 calls either Algorithm 2 or Algorithm 3. The running time of each of these includes:
    - The time to update $\hat{\sigma}$. In Algorithm 2, this includes time $O(L) = O(d)$ to decode the repetition code $\mathcal{L}$ to obtain $\hat{z}$; and then time $O(nn') = O(d)$ to perform the update. In Algorithm 3, the only work is setting $\hat{\sigma}[m] = \perp$ for appropriate values of m, which runs in time $O(n) = O(d)$ as well.
    - The time to decode $\hat{\sigma}$ using $\mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}$. This takes $T_{\mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}}(n)$ time.
    - The time to decode the unary code $\mathcal{U}$ (once for Algorithm 2, twice for Algorithm 3). By Lemma 3, this takes time $O(d)$.
    - The time to call `compute-r` (once for Algorithm 2, twice for Algorithm 3). This takes time $\tilde{O}(d^2)$.
    - Finally, Algorithm 3 picks whichever of the two estimates $\hat{j}_1$ and $\hat{j}_2$ is better. As written in Algorithm 3, this requires computing $\mathrm{Enc}_{\mathcal{G}}(\hat{j}_1)$ and $\mathrm{Enc}_{\mathcal{G}}(\hat{j}_2)$, which naively would include an $O(d^2)$ term in its running time as above. However, the only reason for the $O(d^2)$ term is the time needed to find i given j. In this case, we already have the relevant i (it is the $\hat{i}$ returned by $\mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}$), and so this step can be done in time $O(d) + O(T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}}(n) + n \cdot T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}}(n'))$ as well.
    We note that several times throughout Algorithm 2 and Algorithm 3, the algorithm needs access to $w_{\tilde{i}}$ for some value of $\tilde{i}$; these can be computed in the same way as $\mathrm{Enc}_{\mathcal{G}}(\hat{j}_1)$ above, and so are covered by the $O(d) + O(T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}}(n) + n \cdot T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}}(n'))$ term.

Overall, the decoding complexity is

$$O(n \cdot B) + O\left(n \cdot T_{\mathrm{Dec}_{\mathcal{C}_{\mathrm{in}}}}(n')\right) + O\left(T_{\mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}}(n)\right) + O\left(T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{out}}}}(n) + n \cdot T_{\mathrm{Enc}_{\mathcal{C}_{\mathrm{in}}}}(n')\right) + \tilde{O}(d^2)\,.$$

□

4.2 Analyzing the failure probability of Algorithm 1

Our main result in this section says that the estimate $\hat{j}$ returned by Algorithm 1 is close to j with high probability.

Theorem 2. Fix a constant $p \in (0, 1/2)$. Let $q = 2^{k'}$ for a large enough integer k'. Let $\mathcal{C}_{\text{out}}$ be an $[n, k]_q$ linear code with relative distance δ_{out} that can decode efficiently from e errors and t erasures as long as $2e + t < \delta_{\text{out}} n$.

Let $\mathcal{C}_{\text{in}}$ be an $[n', k']_2$ linear code and suppose that $P_{\text{fail}}^{\mathcal{C}_{\text{in}}} = o(1)$, where the asymptotic notation is as $n' \to \infty$. Let $\mathcal{G} : \{0, \ldots, N-1\} \to \{0,1\}^d$ be the Gray code defined in Definition 7 with $\mathcal{C}_{\text{out}}$ as an outer code and $\mathcal{C}_{\text{in}}$ as an inner code. Suppose that the parameter L in Definition 7 satisfies $L = \omega(\log(kk') \log\log(kk'))$, and suppose that the parameter B in Definition 7 is an absolute constant (independent of k, k', n, n', N). Let $B, \beta, \xi > 0$ be constants so that

$$2\exp(-C_p B)) < \beta < 1/4, \tag{13}$$

where C_p is a constant[8] depending only on p; and

$$2(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}} + 2\beta < \delta_{\text{out}} . \tag{14}$$

Let $j \in \{0, \ldots, N-1\}$ and let $g_j = \text{Enc}_{\mathcal{G}}(j)$. Let $x \in \{0,1\}^d$ be the string $x = g_j \oplus \eta$ where $\eta \sim \text{Ber}(p)^d$ (the result of transmitting g_j through the BSC_p). Let $\hat{j}$ be the output of Algorithm 1 when given as input the string x. Then for sufficiently large t (relative to constants that depend on the constants p, B, β, ξ above),

$$\Pr_{\eta}[|j - \hat{j}| > t] \le \exp(-\Omega(L/\log(kk'))) + \exp(-\Omega(n)) + \exp(-\Omega(t)). \tag{15}$$

Above, we emphasize that the constants inside the $\Omega(\cdot)$ notation in (15) may depend on the constants p, β, B, ξ.

The rest of this section is devoted to the proof of Theorem 2. In each of the following subsections, we analyze a different step of Algorithm 1, and show that it is successful with high probability. Theorem 2 will follow by a union bound over each of these steps; the formal proof of Theorem 2 is at the end of the section.

4.2.1 Estimating the location of the crossover

The purpose of the following claims is to show that, except with probability $\exp(-\Omega(n))$, Algorithm 1 correctly identifies the interval in which the crossover point occurs. Recall the definition of ℓ from (10). Our goal is to show that with high probability, the value $\hat{\ell}$ computed in Line 8 of Algorithm 1 will be close to ℓ.

We start by a simple application of the Chernoff bound (given in Lemma 1) and show that the probability that the majority decoding of a single chunk s_m in Line 3 fails in $\exp(-\Omega(B))$ (assuming that $h_{i,\bar{j}}$ didn't fall in s_m).

Claim 1. Let $\eta \sim \text{Ber}(p)^B$. Then there is some constant $C_p > 0$ so that $\Pr_\eta\left[\text{Maj}\left(1^B \oplus \eta\right) \neq 1\right] = \exp(-C_p \cdot B)$.

[8] The value of C_p is determined in the proof; see Claim 1.

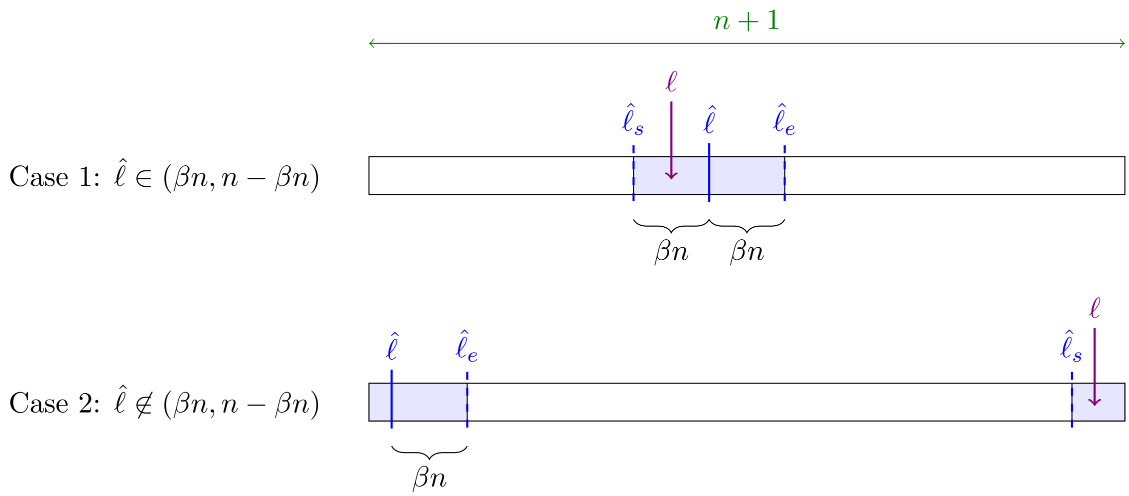


Figure 2: Two cases for where $\hat{\ell}$ can land. As one case see in Case 2, it can be the case that ℓ is in the end of the transmitted codeword whereas $\hat{\ell}$, our estimate of ℓ, is in the beginning.

Proof. The majority fails if at least half of the bits are changed to 0. The expected number of 0s in $1^B + \eta$ is $p \cdot B$. Thus, by Chernoff bound (Lemma 1), the probability that the majority fails can be upper bounded by

$$\exp\left(-\frac{p \cdot B \cdot (\frac{1}{2p}-1)^2}{3}\right) = \exp(-C_p B)$$

where $C_p = p \cdot \left(\frac{1}{2p}-1\right)^2 /3$.

$\square$

Our next focus is to show that $\hat{\ell}$ (computed in line 8 of Algorithm 1), is "close" to ℓ. The next claim considers three possible scenarios depending on the location of $\hat{\ell}$. In the first scenario, $\hat{\ell} \in (\beta n, n-\beta n)$ is "in the middle" of the codeword. In this case, we show that with high probability, $\ell \in [\hat{\ell}_s, \hat{\ell}_e]$ where $\hat{\ell}_s = \hat{\ell} - \beta n$ and $\hat{\ell}_e = \hat{\ell} + \beta n$. The other two cases are that $\hat{\ell} \notin (\beta n, n-\beta n)$ is "in the boundary" of the codeword (with one case for the beginning and one for the end). Here, we show that with high probability $\ell \in [0, \hat{\ell}_e] \cup [\hat{\ell}_s, n]$ where $\hat{\ell}_e$ and $\hat{\ell}_s$ are defined according to lines 1-7 of Algorithm 3 (See also Figure 2). Formally, we have the following claim.

Claim 2. Assume the conditions of Theorem 2. Let $\hat{\ell}$ be the value obtained in line 8 of Algorithm 1. Define the bad event $E_{\hat{\ell}}$ according to the following cases:

1. If $\hat{\ell} \in (\beta n, n-\beta n)$, then $E_{\hat{\ell}}$ is the event that $\ell \notin [\hat{\ell}_s, \hat{\ell}_e]$ where $\hat{\ell}_s = \hat{\ell} - \beta n$ and $\hat{\ell}_e = \hat{\ell} + \beta n$.

2. If $\hat{\ell} \leq \beta n$, then $E_{\hat{\ell}}$ is the event that $\ell \notin [0, \hat{\ell}_e] \cup [\hat{\ell}_s, n]$ where $\hat{\ell}_e = \hat{\ell} + \beta n$ and $\hat{\ell}_s = n+1-(\hat{\ell}-\beta n)$.

3. If $\hat{\ell} \geq n-\beta n$, then $E_{\hat{\ell}}$ is the event that $\ell \notin [0, \hat{\ell}_e] \cup [\hat{\ell}_s, n]$ where $\hat{\ell}_e = \hat{\ell} + \beta n - (n+1)$ and $\hat{\ell}_s = \hat{\ell} - \beta n$.

Then, the probability (over the choice of $\eta \sim \mathrm{Ber}(p)^d$) that $E_{\hat{\ell}}$ occurs is at most $\exp(-\Omega_{B,\beta,p}(n))$.

Proof. We begin with Case 1, namely that $\hat{\ell} \in (\beta n, n-\beta n)$. If $\ell \notin [\hat{\ell}_s, \hat{\ell}_e]$, then $|\ell - \hat{\ell}| > \beta n$. Assume without loss of generality that $\ell > \hat{\ell}$. Let $\hat{s} \in \{0,1\}^{n+1}$ be the quantity computed in Algorithm 1, and suppose that $\hat{\ell} = \mathrm{Dec}_{\mathcal{U}}(\hat{s})$. (Note that $\hat{\ell}$ is either $\mathrm{Dec}_{\mathcal{U}}(\hat{s})$ or $\mathrm{Dec}_{\mathcal{U}^{\mathrm{comp}}}(\hat{s})$; assume without loss of generality that it is $\mathrm{Dec}_{\mathcal{U}}(\hat{s})$, and the other case follows by an identical argument.) By the definition of the unary decoder, it must be that $\Delta(\hat{s}, \mathrm{Enc}_{\mathcal{U}}(\hat{\ell})) \leq \Delta(\hat{s}, \mathrm{Enc}_{\mathcal{U}}(\ell))$. This implies that the number of zeros in $\hat{s}[\hat{\ell} : \ell]$ is greater than the number of ones in this interval. This means that at least $\beta n/2$ values in $\hat{s}$ were decoded incorrectly by the majority decoder. By Claim 1, the probability that a single value of $\hat{s}$ was decoded incorrectly is $\exp(-C_p B)$. Thus, the expected number of values that are decoded incorrectly is $\exp(-C_p B) \cdot (n+1)$. By Lemma 1, as long as $\beta/2 > \exp(-C_p B)$ (which it is by assumption), the probability that $\ell \notin [\hat{\ell}_s, \hat{\ell}_e]$ is at most $\exp(-\Omega(n))$, where the constant in the $\Omega(\cdot)$ depends on B, p and β.

Next, consider Case 2, namely that $\hat{\ell} \leq \beta n$ and that $\ell \notin [0, \hat{\ell}_e] \cup [\hat{\ell}_s, n]$ where $\hat{\ell}_e = \hat{\ell} + \beta n$ and $\hat{\ell}_s = n + 1 - (\hat{\ell} - \beta n)$. Note that in this case it must be that $\ell > \hat{\ell}$ and that $|\ell - \hat{\ell}| > \beta n$. Following the same arguments as in Case 1, we get again that the probability E occurs is $\exp(-\Omega(n))$, and Case 3 follows in the same way. □

Remark 2 (The meaning of "close")**.** We remark that in Cases 2 and 3 Claim 2 it can be the case that ℓ and $\hat{\ell}$ are not close to each other, in the sense that $|\ell - \hat{\ell}|$ is much bigger than βn. This can happen if $\ell \leq \beta n$, but $\hat{\ell} \geq n - \beta n$ or vice versa, as depicted in Figure 2. However, if we consider the values modulo $n+1$, so that the interval $[0, n+1]$ "wraps around," then $\hat{\ell}$ and ℓ are actually close to one another. In this sense, Claim 2 says that ℓ and $\hat{\ell}$ will be "close" to each other with high probability.

4.2.2 Decoding z

In this subsection, we are interested only in the case where $\hat{\ell} \in (\beta n, n - \beta n)$. This is because only Algorithm 2 (not Algorithm 3), attempts to estimate z, and Algorithm 2 is only called when $\hat{\ell}$ is in the middle.[9] In this case, Algorithm 2 decodes the first L bits of x, which should contain the information z_{i+1}. The following claim shows that this decoding process succeeds with probability $\exp(-\Omega(L/\log(kk')))$.

Claim 3. Assume the conditions of Theorem 2. Let $E_{\hat{z}}$ be the bad event that (a) $\hat{\ell} \in (\beta n, n - \beta n)$, and that (b) the quantity $\hat{z}$ computed on Line 3 of Algorithm 2 is incorrect, meaning that $\hat{z} \neq z_{i+1}$. Suppose that the bad event $E_{\hat{\ell}}$ defined in Claim 2 does not occur. Then, conditioned on that, the probability that $E_{\hat{z}}$ occurs is at most

$$\Pr_{\eta}[E_{\hat{z}} \mid \overline{E_{\hat{\ell}}}] \leq \exp(-\Omega(L/\log(kk'))).$$

Proof. Clearly, if $\hat{\ell} \notin (\beta n, n-\beta n)$ the claim trivially holds as the probability of $E_{\hat{z}}$ is 0. Assume that $\hat{\ell} \in (\beta n, n - \beta n)$ and that $E_{\hat{\ell}}$ did not occur. Recall that $\mathcal{L}(z_{i+1})$ simply duplicates each bit of z_{i+1} $L/\log(kk')$ times and that $\tilde{L}$ is a noisy version of $\mathcal{L}(z_{i+1})$. Let us write $\tilde{L} = \tilde{L}_0 \circ \tilde{L}_1 \circ \cdots \circ \tilde{L}_{\log(kk')-1}$, where each $\tilde{L}_m \in \{0,1\}^{L/\log(kk')}$. The decoding algorithm $\mathrm{Dec}_{\mathcal{L}}$ then takes a majority vote of each $\tilde{L}_m$ to recover the estimate $\hat{z}$. Thus, it fails if and only if there is some $m \in [\log(kk') - 1]$ so that $\tilde{L}_m$

[9] Intuitively, the reason that Algorithm 3 does not need to estimate z is because when it is called, the cross-over point is near the boundary. This means that the g_j is already close to a codeword w_i, and we do not need to "translate" the symbols of σ_i into σ_{i+1} or vice versa. Thus, estimating z_{i+1} is not necessary.

has more than half of its $L/\log(kk')$ bits flipped. By the Chernoff bound (Lemma 1), and since the expectation of the number of bits that are flipped is $p \cdot L/\log(kk')$, the probability that this occurs for a particular m most $\exp(-\Omega(L/\log(kk')))$. Applying the union bound over all $\log(kk')$ values of m, the probability that the decoding of $\mathcal{L}(z_{i+1})$ fails is at most $\log(kk') \cdot \exp(-\Omega(L/\log(kk'))) = \exp(-\Omega(L/\log(kk')))$. □

4.2.3 Estimating i

In this subsection, we show that the estimate of $\hat{i}$ obtained in either Algorithm 2 or Algorithm 3 (depending on which was called by Algorithm 1) succeeds with high probability.

In more detail, $\hat{i}$ is computed either in Algorithm 2 or Algorithm 3 based on the location of $\hat{\ell}$ (whether it is in the middle or in the boundary). The following claim shows that with high probability, the estimate of $\hat{i}$ is correct, meaning that $\hat{i} = i$ if $\hat{\ell}$ is in the middle, and $\hat{i}$ is equal to either i or $i+1$ if $\hat{\ell}$ is in the boundary.

Claim 4. Assume the conditions of Theorem 2. Define the bad event $E_{\hat{i}}$ according to the following cases

1. If $\hat{\ell} \in (\beta n, n - \beta n)$ then $E_{\hat{i}}$ is the event that $\hat{i} \neq i$ after performing line 13 in Algorithm 2.
2. If $\hat{\ell} \notin (\beta n, n - \beta n)$ then $E_{\hat{i}}$ is the event that after performing line 11 in Algorithm 3, either
   - $\hat{\ell} \leq \beta n$ and $\hat{i} \neq i$; or
   - $\hat{\ell} \geq n - \beta n$ and $\hat{i} \neq i + 1$.

Assume that neither $E_{\hat{\ell}}$ nor $E_{\hat{z}}$ occurred. Then, conditioned on that, the probability that $E_{\hat{i}}$ occurs is at most

$$\Pr_{\eta}[E_{\hat{i}} \,|\, \overline{E_{\hat{\ell}}}, \overline{E_{\hat{z}}}] \leq \exp(-\Omega(n)),$$

where the constant in the $\Omega(\cdot)$ depends on p, ξ and β defined in (14).

Proof. Based on the location of $\hat{\ell}$, this claim considers the lines 4-13 in Algorithm 2 and lines 8-11 in Algorithm 3. We consider the two cases separately, and show that in each case, before $\hat{i}$ is computed, $\hat{\sigma}$ corresponds to a noisy version of σ_i or σ_{i+1}. Then we can invoke the guarantees of $\mathrm{Dec}_{\mathcal{C}_{\mathrm{out}}}$ to argue that we correctly return either i or $i+1$.

1. **$\hat{\ell}$ is in the middle, i.e., $\hat{\ell} \in (\beta n, n - \beta n)$.**

   Since we assume that $E_{\hat{\ell}}$ did not occur, the quantity ℓ defined in (10) satisfies $\ell \in [\hat{\ell}_s, \hat{\ell}_e]$. This implies that the chunks $\tilde{c}_m$ for $m \in [1, \hat{\ell}_s)$ are corrupted versions of the inner codewords of $c_{i+1}[m], m \in [1, \hat{\ell}_s]$; and that the chunks $\tilde{c}_m$ for $m \in (\hat{\ell}_e, n]$ are corrupted versions of the inner codewords of $c_i[m], m \in (\hat{\ell}_e, n]$.

   First, we argue that with high probability, the chunks $\tilde{c}_m$ are *mostly* correctly decoded to the symbols $\hat{\sigma}[m]$ in Lines 9-11 in Algorithm 1, in the sense that with probability at least $\exp(-\Omega(n))$ over the choice of η, after Line 11, at least a $1 - (1 + \xi) \cdot P_{\mathrm{fail}}^{\mathcal{C}_{\mathrm{in}}}$ fraction of $m \in [1, \hat{\ell}_s) \cup (\hat{\ell}_e, n]$ satisfy

$$\hat{\sigma}[m] = \begin{cases} \sigma_i[m] & m \in (\hat{\ell}_e, n] \\ \sigma_{i+1}[m] & m \in [1, \hat{\ell}_s) \end{cases} \tag{16}$$

where we recall that $\sigma_i \in \mathcal{C}_{\text{out}}$ is the i'th outer codeword, so that $c_i = \sigma_i \circ \mathcal{C}_{\text{in}}$. To see that (16) holds for most m, we observe that for any $m \in [1, \hat{\ell}_s) \cup (\hat{\ell}_e, n]$, the probability that $\hat{\sigma}[m]$ does *not* satisfy (16) is at most $P_{\text{fail}}^{\mathcal{C}_{\text{in}}}$, and so the expected number to not satisfy (16) is $P_{\text{fail}}^{\mathcal{C}_{\text{in}}}(n - 2\beta n - 1)$. Thus, by a Chernoff bound (Lemma 1), the probability that more than $(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}}(n - 2\beta n - 1)$ of these m do not satisfy (16) is at most $\exp(-\Omega(n))$, where the constant inside the $\Omega(\cdot)$ depends on p, β and ξ.

Next, we argue that if the favorable case above occurs, then $\hat{\sigma}$ is a noisy version of σ_i, with not too many errors or erasures.

Since we assume that $E_{\hat{z}}$ did not occur, we have that $\hat{z} = z_{i+1}$. Thus, our choice of ordering of the codewords of $\mathcal{C}$ (see Equation (5)) implies that

$$c_i = c_{i+1} \oplus a_{z_{i+1}} = c_{i+1} \oplus a_{\hat{z}}.$$

Therefore, after $\hat{\sigma}$ is done being updated in Algorithm 2 (that is, after line 12), for the $m \in [1, \hat{\ell}_s) \cup (\hat{\ell}_e, n]$ that satisfy (8), we have

$$\hat{\sigma}[m] = \begin{cases} c_{i+1}[m] \oplus a_{\hat{z}}[m], \text{ if } m \in [1, \hat{\ell}_s) \\ c_i[m], \text{ if } m \in (\hat{\ell}_e, n] \end{cases} = c_i[m]$$

Meanwhile, we have $\hat{\sigma}[m] = \perp$ for all $m \in [\hat{\ell}_s, \hat{\ell}_e]$. In the favorable case that (16) is satisfied for all but $(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}}(n - 2\beta n - 1)$ values of $m \in [1, \hat{\ell}_s) \cup (\hat{\ell}_e, n]$, we conclude that $\hat{\sigma} \in \mathbb{F}_q^n$ is a noisy version of $\sigma_i \in \mathcal{C}_{\text{out}}$, where there are at most $2\beta n + 1$ erasures, and at most $(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}} \cdot (n - 2\beta n - 1)$ errors.

2. **$\hat{\ell}$ is in the boundary, i.e., $\hat{\ell} \notin (\beta n, n - \beta n)$.**

   Again, since we assume that $E_{\hat{\ell}}$ did not occur, the quantity ℓ defined in (10) satisfies $\ell \in [1, \hat{\ell}_e] \cup [\hat{\ell}_s, n]$. If $\ell \in [1, \hat{\ell}_e]$, then for all $m \in (\hat{\ell}_e, \hat{\ell}_s)$, the chunk $\tilde{c}_m$ is a corrupted version of the inner codeword $c_i[m]$. If $\ell \in [\hat{\ell}_s, n]$, then for all $m \in (\hat{\ell}_e, \hat{\ell}_s)$, the chunk $\tilde{c}_m$ is a corrupted version of the inner codeword $c_{i+1}[m]$. By the same argument as in the previous case, we conclude that with probability at least $1 - \exp(-\Omega(n))$ over the choice of η, after Line 11 in Algorithm 1, at least a $1 - (1+\xi) \cdot P_{\text{fail}}^{\mathcal{C}_{\text{in}}}$ fraction of the $m \in (\hat{\ell}_e, \hat{\ell}_s)$ satisfy

   $$\hat{\sigma}[m] = \begin{cases} \sigma_i[m] & \ell \in [1, \hat{\ell}_e] \\ \sigma_{i+1}[m] & \ell \in [\hat{\ell}_s, n] \end{cases}$$

   Since we have $\hat{\sigma}[m] = \perp$ for all $m \in [1, \hat{\ell}_e] \cup [\hat{\ell}_s, n]$, this means that with probability $1 - \exp(-\Omega(n))$, when $\hat{i}$ is computed on Line 11, $\hat{\sigma}$ is a corrupted version of either σ_i or σ_{i+1}, with at most $2\beta n + 1$ erasures and at most $(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}}(n - 2\beta n - 1)$ errors.

Thus, in either case, we have that when $\text{Dec}_{\mathcal{C}_{\text{out}}}$ is called (Line 13 for Algorithm 2 or Line 11 for Algorithm 3), it is called on a corrupted codeword σ that has at most $2\beta n + 1$ erasures and at most $(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}} \cdot (n - 2\beta n - 1)$ errors. Recall that our outer code can recover efficiently from e errors and t erasures, as long as $2e + t < \delta_{\text{out}} n$. Plugging in the number of errors and erasure above, we see that indeed we have

$$\begin{aligned}2e+t &\leq 2(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}}(n-2\beta n-1)+2\beta n+1\\ &\leq \left(2(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}}+2\beta+\frac{1}{n}\right)\cdot n\\ &< \delta_{\text{out}} n\end{aligned}$$

where the last inequality holds for large enough n (relative to $\frac{1}{\delta_{\text{out}}}$) and by our inequality assumption (14). We conclude that with probability at least $1-\exp(\Omega(n))$:

- If $\hat{\ell}$ is in the middle, then $\hat{i}=i$
- If $\hat{\ell}$ is in the boundary, then $\hat{i}$ is either i or $i+1$, depending on which side of the boundary $\hat{\ell}$ is on.

This proves the claim. □

4.2.4 Estimating j

Next, we argue that the estimate $\hat{j}$ that Algorithm 1 returns satisfies $|j-\hat{j}|=\Delta(g_j,g_{\hat{j}})$ with high probability. Before we state and prove that (in Claim 6 below), we first prove the correctness of Algorithm 4, as this is used as a step in the process of deterimining $\hat{j}$.

Claim 5. Algorithm 4 is correct. That is, given $i\in\{0,1,\ldots,q^k-1\}$, $\texttt{compute-r}(i)=r_i$.

Proof. Consider the task of computing r_i from i. From Definition 6, we have

$$r_i=\sum_{t=1}^{i}\Delta(w_{t-1},w_t). \tag{17}$$

Recalling that we may break up the codewords $w_t\in\mathcal{W}$ into chunks, we see that there are three types of contributions to r_i: (1) Contributions from the chunks s_m for $m=1,\ldots,n+1$; (2) contributions from the chunks $\tilde{c}_m$ for $m=1,\ldots,n$; and (3) contributions from the chunks $\tilde{L}$. We consider each of these in turn.

1. **The chunks s_m.** Since $t-1$ and t have different parities, the chunks s_m in w_{t-1} are all completely different from those in w_t. This contributes a total of $(n+1)\cdot B\cdot i$ to the sum in (17).

2. **The chunks $\tilde{c}_m$.** Recall from (5) that $c_t=c_{t-1}+a_{z_t}$, where a_{z_t} is the z_tth row of the generator matrix A of $\mathcal{C}$. Thus, the contribution to (17) of the chunks $\tilde{c}_m$ for $m=1,\ldots,n$ is

$$\sum_{t=1}^{i}\Delta(c_{t-1},c_t)=\sum_{t=1}^{i}\|a_{z_t}\|,$$

where $\|\cdot\|$ denotes hamming weight. For each z, from Observation 2, the number of t so that $z=z_t$ is $\left\lfloor\frac{i+2^z}{2^{z+1}}\right\rfloor$. Thus, the total contribution to r_i from the $\tilde{c}_m$ chunks is

$$\sum_{z=0}^{k'k-1}\lfloor\frac{i+2^z}{2^{z+1}}\rfloor\cdot\|a_z\|.$$

3. **The chunk $\tilde{L}$.** Recall that what goes into the chunk $\tilde{L}$ in c_t is $\mathcal{L}(z_t)$, where $\mathcal{L}$ is the code that repeats each bit representing z_t exactly $L/\log(kk')$ times. Thus, the contribution to (17) from these chunks is

$$\sum_{t=1}^{i} \Delta(\mathcal{L}(z_{t-1}), \mathcal{L}(z_t)) = L/\log(kk') \sum_{t=1}^{i} \Delta(\mathrm{bin}(z_{t-1}), \mathrm{bin}(z_t)),$$

where $\mathrm{bin}(z)$ denotes the binary expansion of z. By Observation 2, $z_t > 0$ if and only if t is even. Thus, for all $z \in \{1, 2, \ldots, kk' - 1\}$, $z = z_t$ implies that $z_{t-1} = z_{t+1} = 0$. This means that the contributions from the two terms

$$\Delta(\mathrm{bin}(z_{t-1}, z_t)) + \Delta(\mathrm{bin}(z_t, z_{t+1}))$$

is given by $2\|\mathrm{bin}(z_t)\|$. Again by Observation 2, the number of times each such z appears as z_t for some $t \le i$ is $\left\lfloor \frac{i+2^z}{2^{z+1}} \right\rfloor$. Thus, the total contribution to (17) of these terms is

$$L/\log(kk') \sum_{z=1}^{kk'-1} \left\lfloor \frac{i + 2^z}{2^{z+1}} \right\rfloor \cdot 2\|\mathrm{bin}(z)\| = L/\log(kk') \sum_{z=0}^{kk'-1} \left\lfloor \frac{i + 2^z}{2^{z+1}} \right\rfloor \cdot 2\|\mathrm{bin}(z)\|$$

where in the equality we have added back in the $t = 0$ term as $\|\mathrm{bin}(0)\| = 0$ and this does not affect the sum.

Finally, we observe that Algorithm 4 exactly computes the three contributions above. First, it initializes $\hat{r}_i$ to $(n+1)Bi$ to account for the s_m chunks; and then it loops over all $z \in \{0, 1, \ldots, kk'-1\}$ and adds the contributions from the $\tilde{c}_m$ and $\tilde{L}$ chunks. □

Claim 6. Let $j \in [N]$ and set i to be such that, $j \in [r_i, r_{i+1})$. Further, let $x = g_j \oplus \eta$ be the noisy version of g_j and $\hat{j}$ be the output of Algorithm 1. Let $t > 0$. If the bad events $E_{\hat{\ell}}, E_{\hat{z}}$, and $E_{\hat{i}}$ do not occur, then the probability that $|j - \hat{j}| > t$, conditional on this, is bounded by

$$\Pr_{\eta}\left[|j - \hat{j}| > t \,|\, \overline{E_{\hat{\ell}}}, \overline{E_{\hat{z}}}, \overline{E_{\hat{i}}} \right] \le \exp(-\Omega(t)) ,$$

where the constant inside the $\Omega(\cdot)$ depends on p.

Proof. As in the proofs of earlier claims, we separate the analysis into two scenarios: one where $\hat{\ell}$ is in the middle, and one where it is in the boundary.

1. $\hat{\ell} \in (\beta n, (1 - \beta)n)$ **is in the middle.**

   In this case the function `get-estimate` (Algorithm 2) is invoked to compute $\hat{j}$. Since we assume that $E_{\hat{\ell}}, E_{\hat{z}}$, and $E_{\hat{i}}$ all hold, we have $\hat{i} = i$. By Claim 5, given i, the value `compute-r`$(\hat{i})$ computed on Line 16 is equal to $r_{\hat{i}}$ and hence equal to r_i.

   Next, we will show that the estimate $\hat{j}$ computed in Algorithm 2 satisfies $\hat{j} \in [r_i, r_{i+1})$. To see this observe that Algorithm 2 first computes $\hat{\bar{j}} = \mathrm{Dec}_{\mathcal{U}}(x[H] \oplus w_{\hat{i}}[H]) = \mathrm{Dec}_{\mathcal{U}}(x[H] \oplus w_i[H])$. Recall from Observation 5 that since $j \in [r_i, r_{i+1})$, we have $g_j[H] \oplus w_i[H] = \mathrm{Enc}_{\mathcal{U}}(\bar{j})$. (Here we are using the fact that the set $H = \{m : w_i[m] \ne w_{i+1}[m]\}$ in Algorithm 2 is the set of elements that appear in the vector h_i). Since $x = g_j \oplus \eta$, we see that

$$\hat{\bar{j}} = \mathrm{Dec}_{\mathcal{U}}(\mathrm{Enc}_{\mathcal{U}}(\bar{j}) \oplus \eta[H]).$$

Now, we consider the probability that $\hat{\bar{j}}$ is very different than $\bar{j}$. For any fixed $\hat{\bar{j}}$, we claim that

$$\Pr_\eta\left[\hat{\bar{j}} = \mathrm{Dec}_{\mathcal{U}}(\mathrm{Enc}_{\mathcal{U}}(\bar{j}) \oplus \eta[H])\right] \le \exp(-\Omega(|\hat{\bar{j}} - \bar{j}|)).$$

Indeed, $\mathrm{Enc}_{\mathcal{U}}(\bar{j})$ and $\mathrm{Enc}_{\mathcal{U}}(\hat{\bar{j}})$ differ on $|\bar{j} - \hat{\bar{j}}|$ coordinates, and $\mathrm{Dec}_{\mathcal{U}}$—which just finds the $\hat{\bar{j}}$ that is closest to the received word—will return $\hat{\bar{j}}$ rather than the correct answer $\bar{j}$ only if at least half of these bits are flipped by $\eta[H]$. The probability that this occurs is the probability that at least half of $|\hat{\bar{j}} - \bar{j}|$ i.i.d. random bits, distributed as $\mathrm{Ber}(p)$, are equal to one. As $p < 1/2$, by a Chernoff bound (Lemma 1), the probablity that this occurs is at most $\exp(-\Omega(|\bar{j} - \hat{\bar{j}}|))$, where the constant inside the $\Omega(\cdot)$ depends on p. Thus, the probability that $\mathrm{Dec}_{\mathcal{U}}$ returns any $\hat{\bar{j}}$ with $|\hat{\bar{j}} - \bar{j}| \ge t$ is at most

$$\begin{aligned}
&\Pr_\eta\left[|\bar{j} - \hat{\bar{j}}| \ge t\right] \\
&\quad = \Pr_\eta\left[|\bar{j} - \mathrm{Dec}_{\mathcal{U}}(\mathrm{Enc}_{\mathcal{U}}(\bar{j}) \oplus \eta[H])| \ge t\right] \\
&\quad \le \sum_{\hat{\bar{j}} \ge \bar{j}+t} \Pr\left[\hat{\bar{j}} = \mathrm{Dec}_{\mathcal{U}}(\mathrm{Enc}_{\mathcal{U}}(\bar{j}) \oplus \eta[H])\right] + \sum_{\hat{\bar{j}} \le \bar{j}-t} \Pr\left[\hat{\bar{j}} = \mathrm{Dec}_{\mathcal{U}}(\mathrm{Enc}_{\mathcal{U}}(\bar{j}) \oplus \eta[H])\right] \\
&\quad \le 2\sum_{s \ge t} \exp(-\Omega(s)) \\
&\quad \le \exp(-\Omega(t)). \qquad (18)
\end{aligned}$$

This shows that $\bar{j}$ is likely close to $\hat{\bar{j}}$. As we observed above, the value $\texttt{compute-r}(\hat{i})$ computed by Algorithm 2 is equal to r_i, so we have

$$\hat{j} = \texttt{compute-r}(\hat{i}) + \hat{\bar{j}} = r_i + \hat{\bar{j}},$$

and by definition we have that

$$j = r_i + \bar{j}.$$

Thus, $|j - \hat{j}| = |\bar{j} - \hat{\bar{j}}|$, and (18) implies that

$$\Pr\left[|j - \hat{j}| \ge t\right] \le \exp(-\Omega(t)) ,$$

as desired.

2. **$\hat{\ell}$ is in the boundary.**

   In this case, the function `get-estimate-boundary` is invoked. Since we assume that $E_{\hat{\ell}}, E_{\hat{z}}$, and $E_{\hat{i}}$, we have that $\hat{i} = i$ if $\ell \in [0, \hat{\ell}_e]$ and $\hat{i} = i + 1$ if $\ell \in [\hat{\ell}_s, n]$.

   First assume that $\ell \in [0, \hat{\ell}_e]$, and note that this implies both that $\hat{i} = i$ and that the crossover point satisfies $h_{i,\bar{j}} \le L + 2\beta n(n' + B)$. Further, by Claim 5, we have $\texttt{compute-r}(\hat{i}) = r_i$.

   Let

   $$H_1 = \{m \mid w_{i+1}[m] \ne w_i[m], 0 \le m \le L + 2\beta n(n' + B)\},$$

   and let

   $$H_2 = \{m \mid w_i[m] \ne w_{i-1}[m], d - 2\beta n(n' + B) \le m \le d\}.$$

(These are the two values of H chosen in Algorithm 3 when computing $\hat{j}_1$ and $\hat{j}_2$, respectively; we give them separate names H_1 and H_2 for the analysis.)

First we analyze the choice of $\hat{j}_1$. By Observation 5 and the fact that $h_{i,\bar{j}} \in H_1$, we have $g_j[H_1] \oplus w_i[H_1] = \mathrm{Enc}_{\mathcal{U}}(\bar{j})$, and so as above we have

$$\hat{\bar{j}}_1 = \mathrm{Dec}_{\mathcal{U}}(\mathrm{Enc}_{\mathcal{U}}(\bar{j}) \oplus \eta[H_1]).$$

The same reasoning as in Case 1 implies that

$$\Pr[|\hat{j}_1 - j| \geq t/2] \leq \exp(-\Omega_p(t)). \tag{19}$$

Further, in this case we also have that

$$|\hat{j}_1 - j| = \Delta(g_j, g_{\hat{j}_1}).$$

Indeed, this follows because, regardless of the noise η, we have $\hat{\bar{j}}_1 \leq |H_1| \leq \Delta(w_i, w_{i+1})$, which means that

$$\hat{j} = \texttt{compute-r}(\hat{i}) + \hat{\bar{j}}_1 = r_i + \hat{\bar{j}}_1 \in [r_i, r_{i+1}).$$

Now, since j and $\hat{j}_1$ are both in the same interval $[r_i, r_{i+1})$, we must have $\Delta(g_j, g_{\hat{j}_1}) = |j - \hat{j}_1|$. This is true because—assuming without loss of generality that $\hat{j}_1 \geq j$—to get from g_j to $g_{\hat{j}_1}$, we flip the bits indexed by $h_i[j+1], h_i[j+2], \ldots, h_i[\hat{j}_1]$, and there are $|j - \hat{j}_1|$ such bits.

Next, we analyze $\hat{j}_2$. First, note that as with $\hat{j}_1$, we have

$$|\hat{j}_2 - j| = \Delta(g_j, g_{\hat{j}_2}). \tag{20}$$

Indeed, by construction we have $\hat{\bar{j}}_2 \leq |H_2|$, which means that $\hat{j}_2 = r_i - \hat{\bar{j}}_2$ is towards the end of the interval $[r_{i-1}, r_i)$; concretely, it implies that the crossover point corresponding to $\hat{j}_2$ satisfies

$$h_{i-1,\hat{j}_2 - r_{i-1}} \in [d - 2\beta n(n' + B), d].$$

Thus, to get from the codeword $g_{\hat{j}_2}$ to the codeword g_j, we need to flip all of the bits indexed by m in the set

$$\{m \mid w_i[m] \neq w_{i-1}[m], h_{i-1,\hat{j}_2 - r_{i-1}} \leq m \leq d\},$$

as well as all the bits indexed by m in the set

$$\{m \mid w_i[m] \neq w_{i+1}[m], 0 \leq m \leq h_{i,\bar{j}}\}.$$

The number of elements in the first set is $\hat{j}_2 = r_i - \hat{\bar{j}}_2$, and the number in the second set is $\bar{j} = j - r_i$. Since $\beta < 1/4$ and $h_{i,\bar{j}} \leq L + 2\beta n(n' + B)$ and $h_{i-1,\hat{j}_2 - r_{i-1}} \geq d - 2\beta n(n' + B)$, these two sets are disjoint. Thus the total number of indices we need to flip to get from $g_{\hat{j}_2}$ to g_j is the sum of the sizes of these two sets, which is

$$(r_i - \hat{\bar{j}}_2) + (j - r_i) = j - \hat{\bar{j}}_2,$$

establishing (20). A similar argument shows that $\Delta(g_{\hat{j}_1}, g_{\hat{j}_2}) = |\hat{j}_1 - \hat{j}_2|$.

Next, note that Algorithm 3 sets $\hat{j} = \hat{j}_2$ only if $\Delta(x, g_{\hat{j}_2}) \le \Delta(x, g_{\hat{j}_1})$. To analyze the probability that this occurs, fix $\hat{j}_1$ and $\hat{j}_2$, and define

$$A_1 := A_1(\hat{j}_1) := \{m \in H_1 \mid h_{i,\bar{j}} \le m \le h_{i,\hat{j}_1 - r_i}\}$$

and let

$$A_2 := A_2(\hat{j}_1, \hat{j}_2) := \{m \in H_2 \mid m \ge h_{i-1,\hat{j}_2 - r_{i-1}}\} \cup \{m \in H_1 \mid m \le \min(h_{i,\bar{j}}, h_{i,\hat{j}_1 - r_i})\}.$$

That is, A_1 is the set of indices m so that $g_{\hat{j}_1}[m] \neq g_j[m]$ but $g_{\hat{j}_2}[m] = g_j[m]$, and A_2 is the set of indices m so that $g_{\hat{j}_2}[m] \neq g_j[m]$ but $g_{\hat{j}_1}[m] = g_j[m]$. Notice that $|A_1| + |A_2| = \Delta(g_{\hat{j}_1}, g_{\hat{j}_2})$. Notice also that

$$|A_1| = \begin{cases} \Delta(g_j, g_{\hat{j}_1}) = \hat{j}_1 - j & j \le \hat{j}_1 \\ 0 & j > \hat{j}_1 \end{cases} \qquad \text{and} \qquad |A_2| = \begin{cases} \Delta(g_j, g_{\hat{j}_2}) = j - \hat{j}_2 & j \le \hat{j}_1 \\ \Delta(g_{\hat{j}_1}, g_{\hat{j}_2}) = \hat{j}_1 - \hat{j}_2 & j > \hat{j}_1 \end{cases} \tag{21}$$

Now, consider the event that Algorithm 3 sets $\hat{j} = \hat{j}_2$, which happens only if $\Delta(x, g_{\hat{j}_2}) \le \Delta(x, g_{\hat{j}_1})$. Note that

$$\Delta(x, g_{\hat{j}_2}) = \Delta(g_j \oplus \eta, g_{\hat{j}_2}) = |A_2| - \|\eta[A_2]\| + \|\eta[\bar{A}_2]\|$$

and a similar expression holds for $\Delta(x, g_{\hat{j}_1})$. Therefore, the event that $\Delta(x, g_{\hat{j}_2}) \le \Delta(x, g_{\hat{j}_1})$ is the same as the event that

$$|A_2| - \|\eta[A_2]\| + \|\eta[A_1]\| \le |A_1| - \|\eta[A_1]\| + \|\eta[A_2]\|,$$

which, rearranging, is the same as the event that

$$|A_2| - 2\|\eta[A_2]\| - (|A_1| - 2\|\eta[A_1]\|) \le 0. \tag{22}$$

Note that $|A_2| - 2\|\eta[A_2]\|$ is a sum of $|A_2|$ independent random variables that are $+1$ with probability $1-p$ and -1 with probability p, and similarly for $|A_1| - 2\|\eta[A_1]\|$. Moreover, since A_1 and A_2 are disjoint, the whole left hand side of (22) is the sum of $|A_1| + |A_2|$ independent ± 1-valued random variables, and the expectation of the left-hand-side is $(|A_2| - |A_1|)(1-2p)$, which is larger than zero when $|A_2| > |A_1|$ and $p < 1/2$. By Hoeffding's inequality (Lemma 2), provided that $|A_2| > |A_1|$ and $p < 1/2$, the probability that (22) occurs is at most

$$\Pr_{\eta}[\Delta(x, g_{\hat{j}_2}) \le \Delta(x, g_{\hat{j}_1})] \le 2 \exp\left(-\Omega_p \left(\frac{(|A_2| - |A_1|)^2}{|A_1| + |A_2|} \right) \right),$$

where the constant inside the $\Omega_p(\cdot)$ depends on the gap between p and $1/2$.

Now, consider the event E that Algorithm 3 picks $\hat{j}_1$ and $\hat{j}_2$ so that all of the following hold:

i. $\Delta(x, g_{\hat{j}_2}) \le \Delta(x, g_{\hat{j}_1})$
ii. $|\hat{j}_1 - j| \le t/2$
iii. $|\hat{j}_2 - j| \ge t$

By a union bound, the probability that E occurs is at most

$$\Pr[E] \le \sum_{\hat{j}_1,\hat{j}_2} \Pr[\Delta(x, g_{\hat{j}_2}) \le \Delta(x, g_{\hat{j}_1})] \le \sum_{\hat{j}_1,\hat{j}_2} 2\exp\left(-\Omega_p\left(\frac{(|A_2|-|A_1|)^2}{|A_1|+|A_2|}\right)\right),$$

where the sum is over all $\hat{j}_1$ and $\hat{j}_2$ that satisfy (ii) and (iii) above. For any such $\hat{j}_1$ and $\hat{j}_2$, we have

$$\frac{(|A_2|-|A_1|)^2}{|A_2|+|A_1|} \ge \frac{|\hat{j}_2 - j|}{6}.$$

Indeed, (ii) and (iii) imply that $|\hat{j}_2 - j| \ge 2|\hat{j}_1 - j|$, so by (21), if $j \ge \hat{j}_1$, then

$$\frac{(|A_2|-|A_1|)^2}{|A_2|+|A_1|} = |A_2| = |\hat{j}_1 - \hat{j}_2| \ge |\hat{j}_2 - j| - |\hat{j}_1 - j| \ge \frac{|\hat{j}_2 - j|}{2},$$

and if $j < \hat{j}_1$, then

$$\frac{(|A_2|-|A_1|)^2}{|A_2|+|A_1|} = \frac{((j-\hat{j}_2)-(\hat{j}_1-j))^2}{\hat{j}_1-\hat{j}_2} \ge \frac{(|\hat{j}_2-j|/2)^2}{(3/2)|j-\hat{j}_2|} = \frac{|\hat{j}_2-j|}{6}.$$

Thus, we have

$$\begin{aligned}
\Pr[E] &\le \sum_{\hat{j}_1,\hat{j}_2} \Pr[\Delta(x, g_{\hat{j}_2}) \le \Delta(x, g_{\hat{j}_1})] \\
&\le \sum_{\hat{j}_1,\hat{j}_2} 2\exp\left(-\Omega_p(|\hat{j}_2 - j|)\right) \\
&\le 2t \sum_{\hat{j}_2} \exp(-\Omega_p(|\hat{j}_2 - j|)) \\
&\le 2t \sum_{s \ge t} \exp(-\Omega_p(s)) \\
&\le 2t\exp(-\Omega_p(t)) = \exp(-\Omega_p(t)).
\end{aligned}$$

Above, we have used the fact from (ii) that there are at most t values of $\hat{j}_1$ in the sum; then we have used (iii) (and the fact that in this case, Algorithm 3 will only choose $\hat{j}_2 < j$) to re-write the sum over $\hat{j}_2$ as a sum over $s \ge t$.

Altogether, by a union bound over the event E analyzed above and the event that $|\hat{j}_1 - j| \ge t/2$ in (19), we conclude that for all t large enough (relative to the gap between p and $1/2$), with probability at least $1 - 2\exp(-\Omega_p(t)) = 1 - \exp(-\Omega_p(t))$, we have both $|\hat{j}_1 - j| \le t/2$ and also that E does not occur. Suppose that this favorable case happens.

Now consider $\hat{j}$. If $\hat{j} = \hat{j}_1$, then by above, $|\hat{j}_1 - j| \le t/2$ and so $|\hat{j} - j| \le t/2$. On the other hand, if $\hat{j} = \hat{j}_2$, then either $|\hat{j} - j| = |\hat{j}_2 - j| < t$, or else event E occurs. (Indeed, if $|\hat{j}_2 - j| \ge j$, then (iii) holds; (i) holds because Algorithm 3 chose $\hat{j} = \hat{j}_2$; and (ii) holds because we are assuming that the favorable case in (19) occurs). But since we are assuming that E does not occur, this implies that $|\hat{j} - j| \le t$. Either way, we conclude $|\hat{j} - j| \le t$ except with probability $\exp(-\Omega_p(t))$, which proves the claim when $\hat{\ell}$ is on the boundary and $\ell \in [0, \hat{\ell}_e]$.

The above handled only the sub-case when $\ell \in [0, \hat{\ell}_e]$. There is also the sub-case where $\ell \in [\hat{\ell}_s, n]$. However, that case follows by an identical argument.

Thus, we have handled both the case when $\hat{\ell}$ is in the middle and the case where $\hat{\ell}$ is on the boundary, and this proves the claim. $\square$

Finally, we are ready to prove Theorem 2.

Proof of Theorem 2. Claim 6 implies that $|\hat{j} - j| \leq t$ with probability at least $1 - \exp(\Omega(t))$, provided that none of $E_{\hat{\ell}}$, $E_{\hat{z}}$, and $E_{\hat{i}}$ occur. By Claims 2, 3 and 4 together with a union bound, the probability that any of these occur is at most $\exp(-\Omega(n)) + \exp(-\Omega(L/\log(kk'))) + \exp(-\Omega(n))$. Thus, we conclude that with probability at least

$$1 - \exp(-\Omega(n)) - \exp(-\Omega(L/\log(kk')) - \exp(-\Omega(t)),$$

we have $|\hat{j} - j| \leq t$, as desired. $\square$

5 Putting all together and choosing parameters

In order to prove Theorem 1, we will plug in a Reed–Solomon (RS) code as $\mathcal{C}_{\text{out}}$. Thus, before we prove the theorem, we recall the definition of RS codes and their basic properties.

Definition 8. Let $\alpha_1, \alpha_2, \ldots, \alpha_n$ be distinct points of the finite field $\mathbb{F}_q$ of order q. For $k < n$ the $[n, k]_q$ *RS code* defined by the evaluation set $\{\alpha_1, \ldots, \alpha_n\}$ is the set of codewords

$$\{(f(\alpha_1), \ldots, f(\alpha_n)) \mid f \in \mathbb{F}_q[x], \deg f < k\} \ .$$

It is well-known that RS codes are Maximum Distance Separable (MDS), which means in particular that an RS code of rate $\mathcal{R}$ and distance δ has

$$\mathcal{R} = 1 - \delta + 1/n.$$

Moreover, encoding and decoding of RS codes can be done in $O(n \cdot \text{poly}(\log n))$ time (see e.g., [Gao03, LCH14]).

We are now ready to prove our main result, which we restate here for the reader's convenience.

Theorem 1. Fix constants $p \in (0, 1/2)$ and a sufficiently small $\varepsilon > 0$. Fix a constant $\mathcal{R} \in (0, 1)$. Let d be sufficiently large, in terms of these constants. Then there is an $n' = \Theta(\log d)$ so that the following holds. Suppose that there exists a binary linear $[n', k']_2$ code $\mathcal{C}_{\text{in}}$ with rate $k'/n' = \mathcal{R}$ so that $\mathcal{C}_{\text{in}}$ has a decoding algorithm $\text{Dec}_{\mathcal{C}_{\text{in}}}$ that has block failure probability on the BSC_p that tends to zero as $n' \to \infty$.[10] Then there is a robust Gray code $\mathcal{G} : \{0, 1, \ldots N - 1\} \to \mathbb{F}_2^d$ and a decoding algorithm $\text{Dec}_{\mathcal{G}} : \mathbb{F}_2^d \to \{0, 1, \ldots N - 1\}$ so that:

1. The rate of $\mathcal{G}$ is $\mathcal{R} - \varepsilon$.
2. Fix $j \in \{0, 1, \ldots N-1\}$, let $\eta \sim \text{Ber}(p)^d$ be a random error vector, and let $\hat{j} := \text{Dec}_{\mathcal{G}}(\mathcal{G}(j) \oplus \eta)$, where $\eta \sim \text{Ber}(p)^d$. Then

$$\Pr_{\eta}[|j - \hat{j}| \geq t] \leq \exp(-\Omega(t)) + \exp\left(-\Omega\left(\frac{d}{\log d}\right)\right) \ ,$$

where the constants inside the $\Omega(\cdot)$ notation depend on p, ε, and $\mathcal{R}$.

[10] See Definition 4 for a formal definition of the failure probability.

3. The running time of $\mathcal{G}$ (the encoding algorithm) is $\tilde{O}(d^3)$ and the running time of $\mathrm{Dec}_{\mathcal{G}}$ (the decoding algorithm) is $\tilde{O}(d^2)$ where the $\tilde{O}(\cdot)$ notation hides logarithmic factors.

Proof of Theorem 1. Given Theorem 2 and Proposition 1 we are left to show that we can choose the outer code and the parameters L, B, β, ξ such that: (i) the rate of our Gray code is $\mathcal{R}_{\text{in}} - \varepsilon$ where $\mathcal{R}_{\text{in}}$ is the rate of the inner code, (ii) that inequalities (13) and (14) hold, (iii) that the running time of our encoder and decoder are as desired.

Let $q = 2^{k'}$ where k' is a large enough integer. Let $\mathcal{C}_{\text{in}}$ be as in the theorem statement. We start with choosing the outer code. We shall use as $\mathcal{C}_{\text{out}}$ an $[n, k]_q$ Reed–Solomon code where the evaluation points are taken to be $\mathbb{F}_q^*$, namely, $n = q - 1$. As Reed–Solomon codes are MDS codes, we have that $\mathcal{R}_{\text{out}} = 1 - \delta_{\text{out}} + \frac{1}{n}$. Set δ_{out} to be $\varepsilon/2$, so $\mathcal{R}_{\text{out}} = 1 - \varepsilon/2 + o(1)$.

Note that as $\mathcal{R}_{\text{in}} = k'/n'$, and $k' = \log q = \log(n+1)$, we have that $n' = (\log(n+1))/\mathcal{R}_{\text{in}}$.

We set $L = (\varepsilon/8)nn'$ and set B to be a sufficiently large constant. Then we plug in n' in the definition of d the length of the encoding of our Gray code (Definition 7) to get,

$$d = n\frac{\log(n+1)}{\mathcal{R}_{\text{in}}} + B \cdot (n+1) + 2L = \Theta(n \log n) ,$$

where the last equality follows as $B = O(1)$ and that $2L = (\varepsilon/4)n((\log(n+1))/\mathcal{R}_{\text{in}})$. We now turn to compute the final rate according to (8). We get

$$\begin{aligned} \mathcal{R}_{\mathcal{G}} &\geq \frac{(1-\delta_{\text{out}}) \cdot \mathcal{R}_{\text{in}}}{1 + (1+\frac{1}{n}) \cdot \frac{1}{\sqrt{n'}} + \frac{\varepsilon}{4}} \\ &\geq \frac{(1-\frac{\varepsilon}{2})\mathcal{R}_{\text{in}}}{1+\frac{\varepsilon}{2}} \\ &\geq \mathcal{R}_{\text{in}} - \varepsilon \end{aligned}$$

where the first inequality follows as there exists a large enough integer n for which $(1+1/n)/\sqrt{n'} < \varepsilon/4$ (recall that $n' = O(\log n)$).

Now, note that by our choice of L, the failure probability of our algorithm is

$$\Pr\left[|j - \hat{j}| \geq t\right] \leq \exp(-\Omega(t)) + \exp(-\Omega(n)) .$$

We now show how to get the final failure probability as a function of d. Recall that $d = \Theta(n \cdot \log n)$, which implies that $n = \Theta(d/\log d)$. Indeed, let C_1, C_2 be constants such that $d/C_1 \leq n \log n \leq d/C_2$ for large enough n. It holds that

$$n \geq \frac{d}{C_1 \log n} \geq \frac{d}{C_1 \log d - \log(C_2 \log n)} \geq \frac{d}{C_1 \log d} ,$$

and a similar computation also shows that $n = O(d/\log d)$. Thus,

$$\Pr[|j - \hat{j}| \geq t] \leq \exp(-\Omega(t)) + \exp\left(-\Omega\left(\frac{d}{\log d}\right)\right) .$$

We proceed to show that the conditions given in Theorem 2 indeed hold.

First, we observe that our choice of L indeed satisfies $L = \omega(\log(kk')\log\log(kk'))$, as $L = \Theta(\varepsilon n \log n)$, which is much larger. Next, we show that we can choose constants β, ξ so that (13) and (14) hold, namely that

$$2\exp(-C_p B) < \beta < 1/4 \qquad \text{and} \qquad 2(1+\xi)P_{\text{fail}}^{\mathcal{C}_{\text{in}}} + 2\beta < \delta_{\text{out}} .$$

First, we choose a positive constant $\beta < \min\{1/4, \frac{\delta_{\text{out}}}{4}\}$, recalling that $\delta_{\text{out}} = 1 - \mathcal{R}$ for our Reed-Solomon code $\mathcal{C}_{\text{out}}$, and thus δ_{out} is also a constant. Thus the second inequality in (13) is satisfied. Next, we note that by assumption, $P_{\text{fail}}^{\mathcal{C}_{\text{in}}} = o(1)$ as $n' \to \infty$, and thus for large enough values of n', we have $P_{\text{fail}}^{\mathcal{C}_{\text{in}}} < \delta_{\text{out}}/8$; then we can choose any $\xi < 1$ and satisfy (14) given that $\beta < \delta_{\text{out}}/4$. Finally, we may choose B to be a sufficiently large constant (larger than $\ln(2/\beta)/C_p$) and the first inequality in (13) will hold as well.

Finally, we analyze the final running time of our scheme given our choice of the outer code. Since RS codes can be encoded in time $O(n \cdot \text{poly}(\log n))$, plugging this in Proposition 1 and noting that $n' = \log(n+1)/\mathcal{R}_{\text{in}} \leq \log(d)/\mathcal{R}_{\text{in}}$, we get that the encoding of Gray code can be done in time

$$\tilde{O}(d^3) + O(d \cdot \text{poly}(\log d)) + O(d \cdot T_{\text{Enc}_{\mathcal{C}_{\text{in}}}}(\log(d)/\mathcal{R}_{\text{in}})) = \tilde{O}(d^3) ,$$

where the last equality follows since $\mathcal{C}_{\text{in}}$ can be encoded in polynomial time. Further, the decoding time of RS codes is also $O(n \cdot \text{poly}(\log n))$, and thus, by our choice of B and the fact that $n' = O(\log n)$, we get that the decoding time is

$$\tilde{O}(d^2) + O(d \cdot \text{poly}(\log d)) + O\left(d \cdot T_{\text{Dec}_{\mathcal{C}_{\text{in}}}}(n')\right) = \tilde{O}(d^2) .$$

The last equality above follows because, without loss of generality, we may assume that $T_{\text{Dec}_{\mathcal{C}_{\text{in}}}}(n')$ is at most $\text{poly}(n') \cdot 2^{k'}$, the running time of the brute-force maximum-likelihood decoder. As we have $k' = \log(n+1)$, this is $n \cdot \text{polylog}(n) = d \cdot \text{polylog}(d)$, and hence the final term above is $\tilde{O}(d^2)$. □

Acknowledgements

We thank the Simons Institute for Theoretical Computer Science for their hospitality and support.

References

[ACL+21] Jayadev Acharya, Clement Canonne, Yuhan Liu, Ziteng Sun, and Himanshu Tyagi. Distributed estimation with multiple samples per user: Sharp rates and phase transition. *Advances in neural information processing systems*, 34:18920–18931, 2021.

[ALP21] Martin Aumüller, Christian Janos Lebeda, and Rasmus Pagh. Differentially private sparse vectors with low error, optimal space, and fast access. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 1223–1236, 2021.

[ALS23] Jayadev Acharya, Yuhan Liu, and Ziteng Sun. Discrete distribution estimation under user-level local differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 8561–8585. PMLR, 2023.

[Ari08] Erdal Arikan. A performance comparison of polar codes and reed-muller codes. *IEEE Communications Letters*, 12(6):447–449, 2008.

[AS23] Emmanuel Abbe and Colin Sandon. A proof that reed-muller codes achieve shannon capacity on symmetric channels. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 177–193. IEEE, 2023.

[BGN+22] Jarosław Błasiok, Venkatesan Guruswami, Preetum Nakkiran, Atri Rudra, and Madhu Sudan. General strong polarization. *ACM Journal of the ACM (JACM)*, 69(2):1–67, 2022.

[FW24] Dorsa Fathollahi and Mary Wootters. Improved construction of robust gray code. *arXiv preprint arXiv:2401.15291*, 2024.

[Gao03] Shuhong Gao. A new algorithm for decoding reed-solomon codes. In *Communications, information and network security*, pages 55–68. Springer, 2003.

[Gra53] Frank Gray. Pulse code communication, March 17 1953. US Patent 2,632,058.

[GRY20] Venkatesan Guruswami, Andrii Riazanov, and Min Ye. Arikan meets shannon: Polar codes with near-optimal convergence to channel capacity. In *Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing*, pages 552–564, 2020.

[GW24] Venkatesan Guruswami and Hsin-Po Wang. Capacity-Achieving Gray Codes, 2024. arXiv eprint forthcoming.

[GX14] Venkatesan Guruswami and Patrick Xia. Polar codes: Speed of polarization and polynomial gap to capacity. *IEEE Transactions on Information Theory*, 61(1):3–16, 2014.

[HAU14] Seyed Hamed Hassani, Kasra Alishahi, and Rüdiger L Urbanke. Finite-length scaling for polar codes. *IEEE Transactions on Information Theory*, 60(10):5875–5898, 2014.

[HVDH21] David Harvey and Joris Van Der Hoeven. Integer multiplication in time $O(n \log n)$. *Annals of Mathematics*, 193(2):563–617, 2021.

[Knu11] Donald E Knuth. *The art of computer programming, volume 4A: combinatorial algorithms, part 1*. Pearson Education India, 2011.

[LCH14] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S Han. Novel polynomial basis and its application to reed-solomon erasure codes. In *2014 ieee 55th annual symposium on foundations of computer science*, pages 316–325. IEEE, 2014.

[LP24] David Rasmussen Lolck and Rasmus Pagh. Shannon meets gray: Noise-robust, low-sensitivity codes with applications in differential privacy. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1050–1066. SIAM, 2024.

[MU17] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[RP23] Galen Reeves and Henry D Pfister. Reed–muller codes on bms channels achieve vanishing bit-error probability for all rates below capacity. *IEEE Transactions on Information Theory*, 2023.

[SK20] Madhu Sudan and Kenz Kallal. Essential Coding Theory Lecture Notes, Lecture 3, 2020. Available at: https://people.seas.harvard.edu/ madhusudan/courses/Spring2020/scribe/lect03.pdf. Accessed June 2024.

[TV13] Ido Tal and Alexander Vardy. How to construct polar codes. *IEEE Transactions on Information Theory*, 59(10):6562–6582, 2013.

[WX10] Wenjie Wang and Xiang-Gen Xia. A closed-form robust chinese remainder theorem and its performance analysis. *IEEE Transactions on Signal Processing*, 58(11):5655–5666, 2010.

[XXW20] Li Xiao, Xiang-Gen Xia, and Yu-Ping Wang. Exact and robust reconstructions of integer vectors based on multidimensional chinese remainder theorem (md-crt). *IEEE Transactions on Signal Processing*, 68:5349–5364, 2020.