Improved Construction of Robust Gray Codes

Dorsa Fathollahi
Department of Electrical Engineering
Stanford University
Stanford, CA, USA
Email: dorsafth@stanford.edu

Mary Wootters
Departments of Computer Science and Electrical Engineering
Stanford University
Stanford, CA, USA
Email: marykw@stanford.edu

2024 IEEE International Symposium on Information Theory (ISIT) | 979-8-3503-8284-6/24/$31.00 ©2024 IEEE | DOI: 10.1109/ISIT57864.2024.10619585

***Abstract*—A *robust Gray code*, formally introduced by (Lolck and Pagh, SODA 2024), is a Gray code that additionally has the property that, given a noisy version of the encoding of an integer j, it is possible to reconstruct $\hat{j}$ so that $|j-\hat{j}|$ is small with high probability. That work presented a transformation that transforms a binary code $\mathcal{C}$ of rate R to a robust Gray code with rate $\Omega(R)$, where the constant in the $\Omega(\cdot)$ can be at most $1/4$. We improve upon their construction by presenting a transformation from a (linear) binary code $\mathcal{C}$ to a robust Gray code with similar robustness guarantees, but with rate that can approach $R/2$.**

A full version of this paper can be found in [1].

I. Introduction

In [2], Lolck and Pagh introduce the notion of a **robust Gray code**. Informally, a robust Gray code $\mathcal{G} \subseteq \{0,1\}^d$ has an encoding map $\mathrm{Enc}_{\mathcal{G}} : \{0,\dots,N-1\} \to \{0,1\}^d$ that maps integers to bitstrings, with the following desiderata.

- $\mathcal{G}$ **should be a Gray code.**[1] That is, for any $j \in \{0,\dots,N-2\}$, $|\mathrm{Enc}_{\mathcal{G}}(j) - \mathrm{Enc}_{\mathcal{G}}(j+1)| = 1$.
- $\mathcal{G}$ **should be "noise robust."** Informally, this means that we should be able to approximately recover an integer $j \in \{0,\dots,N-1\}$ given a noisy version of $\mathrm{Enc}_{\mathcal{G}}(j)$. Slightly more formally, $\mathcal{G}$ should have a decoding map $\mathrm{Dec}_{\mathcal{G}} : \{0,1\}^d \to \{0,\dots,N-1\}$, so that when $\eta \sim \mathrm{Ber}(p)^n$, the estimate $\hat{j} = \mathrm{Dec}_{\mathcal{G}}(\mathrm{Enc}_{\mathcal{G}}(j) \oplus \eta)$ should be close to j with high probability.
- $\mathcal{G}$ **should have high rate.** The rate $\frac{\log N}{d}$ of $\mathcal{G}$ should be as close to 1 as possible.
- $\mathcal{G}$ **should have efficient algorithms.** Both $\mathrm{Enc}_{\mathcal{G}}$ and $\mathrm{Dec}_{\mathcal{G}}$ should have running time polynomial (ideally, near-linear) in d.

Robust Gray codes have applications in differential privacy; see [2]–[5] for more details on the connection. It is worth mentioning that there exist non-binary codes based on the Chinese Remainder Theorem [6], [7] that have nontrivial sensitivity, but in our work, we focus on binary codes.

Our Contributions. In this paper, we improve upon the construction of [2] by giving a construction of a robust Gray code with the same robustness guarantees, but better rate.

More precisely, for $p \in (0, 1/2)$, [2] give a general recipe for turning a good error-correcting code $\mathcal{C}$ with rate R into a robust Gray code $\mathcal{G}$ with rate $\Omega(R)$, and with the following robustness guarantee:

$$\Pr_{\eta\sim\mathrm{Ber}(p)^d}[|j - \mathrm{Dec}_{\mathcal{G}}(\mathrm{Enc}_{\mathcal{G}}(j)+\eta)| \geq t] \leq \exp(-\Omega(t)) + \exp(-\Omega(d)) + O(P_{\text{fail}}(\mathcal{C})), \quad (1)$$

where $P_{\text{fail}}(\mathcal{C})$ is the failure probability of the code $\mathcal{C}$ on the binary symmetric channel with parameter p:

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

Our main result is a similar transformation that turns a (linear) binary code $\mathcal{C}$ with good performance on the binary symmetric channel into a robust Gray code $\mathcal{G}$. We obtain a similar robustness guarantee as (1) (see Theorem 1 for the precise statement), but with better rate. Concretely, if the original code $\mathcal{C}$ has rate $R \in (0,1)$, the rate of the robust Gray code from [2] is proven to be $\Omega(R)$, where the constant inside the Ω approaches $1/4$ when $\mathcal{C}$ has sublinear distance; this comes from the fact that the a codeword in their final construction involves *four* codewords from $\mathcal{C}$. In contrast, under the same conditions, our robust Gray code $\mathcal{G}$ has rate approaching $R/2$; this is because our construction involves only *two* codewords from $\mathcal{C}$. (See Observation 2 for the formal statement). Moreover, if the encoding and decoding algorithms for $\mathcal{C}$ are efficient, then so are the encoding and decoding algorithms for our construction $\mathcal{G}$; concretely, the overhead on top of the encoding and decoding algorithms for $\mathcal{C}$ is $O(d)$ (see Lemma 3 for the precise statement).

As a result, when instantiated with, say, a constant-rate Reed-Muller code or a polar code (both of which have sublinear distance and good performance on the BSC(p) (see, e.g., [8]–[12])), our construction gives efficient robust Gray codes with a rate about two times larger than than previous work, approaching $R/2$ if $\mathcal{C}$ has rate R.

Main Idea. The idea of our transformation is quite simple, and follows the same high-level structure as [2]. We begin with our base code $\mathcal{C}$, and use it to construct an intermediate code $\mathcal{W}$ (with an appropriate ordering). Then we add new codewords to $\mathcal{W}$ to complete it to a Gray code. For example, if w_i, w_{i+1} are two consecutive codewords in $\mathcal{W}$, then we will insert $\Delta(w_i, w_{i+1}) - 1$ codewords in between them, iteratively flipping bits to move from w_i to w_{i+1}.

[1] The paper [2] 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).

The main difference between our construction and that of previous work is how we build and order $\mathcal{W}$. First, we use a standard Gray code to construct an *ordering* of the codewords in $\mathcal{C}$. Then, we build $\mathcal{W}$ as follows. Let c_i be the i'th codeword in $\mathcal{C}$. Then the i'th codeword in $\mathcal{W}$ is given by

$$w_i = s_i \circ c_i \circ s_i \circ c_i \circ s_i,$$

where s_i is a short string that is all zeros if i is even and all ones otherwise, and $\circ$ denotes concatenation. Then we form $\mathcal{G}$ by interpolating as described above.

Our decoding algorithm ends up being rather complicated, but the idea is simple. Suppose that for a codeword $g \in \mathcal{G}$, we see a corrupted version $g + \eta \in \mathbb{F}_2^d$, where η is a noise vector. As described above, g is made up of a prefix from w_{i+1} and a suffix from w_i, for some i. Let $h \in [d]$ be the index where g "crosses over" from w_{i+1} to w_i. Notice that, as this crossover point can only be in one place, at least one of the two codewords of $\mathcal{C}$ appearing in g will be complete, and equal to either c_i or c_{i+1}. Thus, if we could identify where the crossover point h was, then we could use $\mathcal{C}$'s decoder to decode whichever the complete $\mathcal{C}$-codeword was to identify i; and then use our knowledge of where h is to complete the decoding. The simple observation behind our construction is that, because the strings s_i (which are either all zeros or all ones) flip with the parity of i, we *can* tell (approximately) where h was! Indeed, these strings will be all zeros before h and all ones after h, or vice versa. Of course, some noise will be added, but provided that the length of the strings s_i are long enough, we will still be able to approximately locate h with high probability.

However, there are several challenges to implementing this simple idea. For example, given i and h, how do we efficiently compute j? (Here is where the fact that we ordered $\mathcal{C}$ carefully comes in; it's not trivial because the number of codewords of $\mathcal{G}$ inserted between w_i and w_{i+1} depends on i, so naively adding up the number of codewords of $\mathcal{G}$ that come before w_i and then adding h would take exponential time.) Or, what happens when the crossover point h is very close to the end of $g_j + \eta$? (Here, it might be the case that we mis-identify i; but we show that this does not matter, with high probability, because our final estimate will still be close to j with high probability). In the rest of the paper, we show how to deal with these and other challenges.

II. Preliminaries

We begin by setting notation. Throughout, we work with linear codes over $\mathbb{F}_2$, so all arithmetic between codewords is modulo 2. For $x, y \in \mathbb{F}_2^\ell$, let $\Delta(x, y)$ denote the Hamming distance between x and y. We use $\|x\|$ to denote the Hamming weight of a vector $x \in \mathbb{F}_2^\ell$. For a code $\mathcal{C} \subseteq \mathbb{F}_2^n$, the minimum distance of the code is given by $D(\mathcal{C}) := \min_{c \neq c' \in C} \Delta(c, c')$.

For two strings s_1 and s_2, we use $s_1 \circ s_2$ to denote the concatenation of s_1 and s_2. For a string $s \in \mathbb{F}_2^\ell$ and for $i \leq \ell$, we use $\mathrm{pref}_i(s) \in \mathbb{F}_2^i$ to denote the prefix of the string s ending at (and including) index i. Analogously, we use $\mathrm{suff}_i(s) \in \mathbb{F}_2^{\ell - i}$ be defined as the suffix of s starting at (and including) index i. For an integer ℓ, we use $[\ell]$ to denote the set $\{1, \ldots, \ell\}$.

For $\ell \in \mathbb{Z}$, let $\mathrm{Maj}_\ell : \mathbb{F}_2^\ell \to \mathbb{F}_2$ be majority function on ℓ bits. (In the case that ℓ is even and a string $y \in \mathbb{F}_2^\ell$ has an equal number of zeros and ones, $\mathrm{Maj}_\ell(y)$ is defined to be a randomized function that outputs 0 or 1 each with probability $1/2$.) We use $\mathrm{Ber}(p)$ to denote the Bernoulli-p distribution on $\mathbb{F}_2$, so if $X \sim \mathrm{Ber}(p)$, then X is 1 with probability p and 0 with probability $1 - p$.

Next we define Binary Reflected Codes, a classical Gray code ([13]; see also, e.g., [14]); we will use these to define our ordering on $\mathcal{C}$. 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 $i \in \{0, \ldots, 2^k - 1\}$,
   - If $i < 2^{k-1}$, then $\mathcal{R}_k(i) = 0 \circ R_{k-1}(i)$
   - If $i \geq 2^{k-1}$, then $\mathcal{R}_k(i) = 1 \circ \overline{R}_{k-1}(2^{k-1} - (i - 2^{k-1}) - 1) = 1 \circ \overline{R}_{k-1}(2^k - i - 1)$.

It is not hard to see that $\mathcal{R}_k$ is indeed a Gray code.

As an example, $R_2(1) = 0 \circ R_1(1) = 01$, as $1 < 2^{k-1} = 2$; and $R_2(2) = 1 \circ R_1(1) = 11$, as $2 \geq 2^{k-1} = 2$.

We will need one more building-block, the **Unary code.** The Unary code $\mathcal{U} \subseteq \mathbb{F}_2^\ell$ is defined as the image of the encoding map $\mathrm{Enc}_\mathcal{U} : \{0, \ldots, \ell\} \to \mathbb{F}_2^\ell$ given by $\mathrm{Enc}_\mathcal{U}(v) := 1^v \circ 0^{\ell - v}$. The decoding map $\mathrm{Dec}_\mathcal{U} : \mathbb{F}_2^\ell \to \{0, \ldots, \ell\}$ is given by

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

III. Construction

We recall the high-level overview of our construction from the introduction: To construct $\mathcal{G}$ we will start with a base code $\mathcal{C}$ where $\mathcal{C} \subseteq \mathbb{F}_2^n$, which we will order in a particular way (Definition 1). Then we construct an intermediate code $\mathcal{W} = \{w_0, \ldots, w_{2^k - 1}\} \subseteq \mathbb{F}_2^d$ by transforming the codewords of $\mathcal{C}$ (Definition 2); the codewords of $\mathcal{W}$ inherit an order of $\mathcal{C}$. Finally, we create final code $\mathcal{G} \subseteq \mathbb{F}_2^d$ by adding new codewords that "interpolate" between the codewords of $\mathcal{W}$ so that it satisfies the Gray code condition (Definition 5). We discuss each of these steps more below.

First, given a base code $\mathcal{C} \subset \mathbb{F}_2^n$, we define an ordering on the elements of $\mathcal{C}$ as follows.

Definition 1. Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a linear code with block length n and dimension k. Let $A_\mathcal{C} \in \mathbb{F}_2^{k \times n}$ be a generator matrix for $\mathcal{C}$, and let a_z denote the z-th row of $A_\mathcal{C}$. Given $i \in \{0, \ldots, 2^k - 1\}$, define z_i to be the unique integer so that $\mathcal{R}_k(i)[z_i] \neq \mathcal{R}_k(i+1)[z_i]$. Let $c_0 = 0^n$. Then, for all $i \in \{1, \ldots, 2^k - 1\}$, the i-th codeword of $\mathcal{C}$ is defined by $c_i = c_{i-1} + a_{z_i}$.

In the full version of the paper, we prove that indeed this ordering hits all of the codewords. For now, we move on to define our intermediate code $\mathcal{W}$.

Definition 2. Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a linear code of dimension k. Let $(c_0, \ldots, c_{2^k - 1})$ denote the ordering of codewords in $\mathcal{C}$ as per Definition 1. Let $d = 2n + 3D(\mathcal{C})$. The intermediate code

$\mathcal{W}$, along with its ordering, is defined as follows. For each $i \in \{0, \ldots, 2^k - 1\}$, define $w_i \in \mathbb{F}_2^d$ by the equation

$$w_i = \begin{cases} 0^{D(\mathcal{C})} \circ c_i \circ 0^{D(\mathcal{C})} \circ c_i \circ 0^{D(\mathcal{C})} & \text{if } i \text{ is even} \\ 1^{D(\mathcal{C})} \circ c_i \circ 1^{D(\mathcal{C})} \circ c_i \circ 1^{D(\mathcal{C})} & \text{if } i \text{ is odd} \end{cases} \quad (2)$$

Then, $\mathcal{W} \subseteq \mathbb{F}_2^d$ is defined by $\mathcal{W} = \{w_i : i \in \{1, \ldots, 2^k - 1\}\}$, where $\mathcal{W}$ is ordered as $(w_0, \ldots, w_{2^k-1})$.

Definition 3. For every i, define $r_i = \sum_{\ell=1}^{i} \Delta(w_{\ell-1}, w_\ell)$.

It will be useful to be able to locate $j \in [N]$ within a block $[r_i, r_{i+1})$. To that end, we introduce the following notation.

Definition 4. Let $j \in \{0, \ldots, N-1\}$. Let $i \in \{0, \ldots, 2^k - 1\}$ be such that $j \in [r_i, r_{i+1})$. Then we will use the notation $\bar{j}$ to denote $j - r_i$. That is, $\bar{j}$ is the index of j in the block $[r_i, r_{i+1})$.

Finally, we to create our robust Gray code $\mathcal{G}$, given any two consecutive codewords in $\mathcal{W}$, we inject extra codewords between them as follows.

Definition 5 (Definition of our robust Gray code $\mathcal{G}$; and the parameters $r_i, h_{i,j}$)**.** Let $\mathcal{W} \subseteq \mathbb{F}_2^d$ be a code defined as in Definition 2. For each $i \in [2^k]$, recall that $r_i = \sum_{\ell=1}^{i} \Delta(w_{\ell-1}, w_\ell)$ according to Definition 3, and let $N = r_{2^k}$. For $i \in [2^k]$ and $1 \le \bar{j} < \Delta(w_i, w_{i+1})$, let $h_{i,\bar{j}} \in \{0, \ldots, d-1\}$ be the $\bar{j}$-th index where codewords w_i and w_{i+1} differ.

Define $g_0 = w_0$. Fix $j \in \{1, \ldots, N-1\}$. Find i such that $j \in [r_i, r_{i+1})$. (Note that this can be done efficiently, via binary search.) If $j = r_i$, we define $g_j \in \mathbb{F}_2^d$ by $g_j = w_i$. On the other hand, if $j \in (r_i, r_{i+1})$, then we define $g_j \in \mathbb{F}_2^d$ as

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

Finally, define $\mathcal{G} \subseteq \mathbb{F}_2^d$ by $\mathcal{G} = \{g_i : i \in \{0, \ldots, N-1\}\}$, along with the encoding map $\text{Enc}_{\mathcal{G}} : \{0, \ldots, N-1\} \to \mathbb{F}_2^d$ given by $\text{Enc}_{\mathcal{G}}(i) = g_i$.

We will also define $h_i = (h_{i,1}, h_{i,2}, \ldots, h_{i,\Delta(w_i,w_{i+1})-1}) \in \{0, \ldots, d-1\}^{\Delta(w_i,w_{i+1})-1}$ to be the vector of all indices in which w_i and w_{i+1} differ, in order, except for the last one.[2]

Note that, in this notation, 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 $h_{i,\bar{j}}$.

It follows from Definition 5 that $\mathcal{G}$ is indeed a Gray code, and that it has rate approaching $R/2$. Formally, we have the following two observations.

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

Observation 2. Suppose that $\mathcal{C} \subseteq \mathbb{F}_2^n$ has rate $R = \log_2 |\mathcal{C}|/n$ and distance $o(n)$. Then the code $\mathcal{G}$ constructed as in Definition 5 has rate that approaches $R/2$ as $N \to \infty$.

In the full version, we prove that $\mathcal{G}$ (Definition 5) is indeed an injective map, so we don't need to worry about running into the same string multiple times while "interpolating."

[2]The reason we don't include the last one is because once the last differing bit has been flipped, g_j will lie in $[w_{i+1}, w_{i+2})$, not $[w_i, w_{i+1})$.

IV. Decoding Algorithm and Analysis

In this section, we define our decoding algorithm and analyze it. We begin with some notation for the different parts of the codewords $g_j \in \mathcal{G}$. For a string x, we use $x[i : i']$ to denote the substring $(x_i, x_{i+1}, \ldots, x_{i'-1})$. Moreover, $[i :]$ denotes the suffix of string x staring at i. With this notation, for any $x \in \mathbb{F}_2^d$, define the following substrings:

- $s_1(x) = x[0 : D(\mathcal{C})]$
- $\tilde{c}_1(x) = x[D(\mathcal{C}) : D(\mathcal{C}) + n]$
- $s_2(x) = x[D(\mathcal{C}) + n : 2D(\mathcal{C}) + n]$
- $\tilde{c}_2(x) = x[2D(\mathcal{C}) + n, 2D(\mathcal{C}) + 2n]$
- $s_3(x) = x[2D(\mathcal{C}) + 2n : 3D(\mathcal{C}) + 2n]$.

Notice that if $x \in \mathcal{G}$, then $\tilde{c}_1$ and $\tilde{c}_2$ are in locations corresponding to the codewords of $\mathcal{C}$ that appear in codewords of $\mathcal{W}$, while s_1, s_2, and s_3 are in locations corresponding to the $0^{D(\mathcal{C})}$ and $1^{D(\mathcal{C})}$ strings.

Before we formally state the algorithm (Algorithm 2 below), introduce some notation and state a lemma to motivate its structure.

First, it is not hard to see (and we prove formally in the full version) that if we write $g = s_1 \circ \tilde{c}_1 \circ s_2 \circ \tilde{c}_2 \circ s_3$, then at most one of these "chunks" are broken up by the "crossover point" $h_{i,\bar{j}}$; the other four are equal to the corresponding substring in w_i or w_{i+1}. We say that a substring in $\mathcal{S}$ that is equal to its corresponding substring in w_i or w_{i+1} is a **full chunk**. Thus, there are at least four full chunks in any $g_j \in \mathcal{G}$. Notice that it is possible that a substring $\tilde{c}_\ell$ is in $\mathcal{C}$ but is not a full chunk. We say that **all full chunks are decoded correctly** if, for full chunk of x, when we run the corresponding decoder, we get the right answer. That is, if $\tilde{c}_1(x)$ is a full chunk, then if we were to run $\text{Dec}_{\mathcal{C}}$ on $\tilde{c}_1(x)$ we would obtain $\tilde{c}_1(g_j)$, and similarly for $\tilde{c}_2$; and if $s_1(x)$ is a full chunk, and we were to run $\text{Maj}_{D(\mathcal{C})}$ on $s_1(x)$, we would obtain $s_1(g_j)$, and similarly for s_2 and s_3.

The intuition behind our algorithm, as mentioned above, is that we can detect (approximately) where the "crossover point" $h_{i,\bar{j}}$ is by seeing what the chunks $s_1(x), s_2(x)$ and $s_3(x)$ decode to under majority-vote. This motivates three cases that will be reflected in our algorithm, which we characterize with the following lemma.

Lemma 1. Let $g_j \in \mathcal{G}$ and let i be such that $j \in [r_i, r_{i+1})$. Let $\eta \sim \text{Ber}(p)^d$ where $p \in (0, 1/2)$. Let $x = g_j + \eta$ be a received input. Then define $\hat{v}_{i'} = \text{Dec}_{\mathcal{C}}(\tilde{c}_{i'}(x))$ for $i' \in \{1, 2\}$ and $b_{i'} = \text{Maj}_{D(\mathcal{C})}(s_{i'}(x))$ for $i' \in \{1, 2, 3\}$. Assume that all full chunks are decoded correctly by their corresponding decoder. Then the following hold.

1) If $(b_1, b_2, b_3) \in \{(1, 1, 0), (0, 0, 1)\}$, then $\text{Enc}_{\mathcal{C}}(\hat{v}_1) = c_{i+1}$ and $h_{i,\bar{j}} \ge n + D(\mathcal{C})$.
2) If $(b_1, b_2, b_3) \in \{(0, 1, 1), (1, 0, 0)\}$, then $\text{Enc}_{\mathcal{C}}(\hat{v}_2) = c_i$ and $h_{i,\bar{j}} \le n + 2D(\mathcal{C})$.
3) If $(b_1, b_2, b_3) \in \{(0, 0, 0), (1, 1, 1)\}$, then $\text{Enc}_{\mathcal{C}}(\hat{v}_1) = \text{Enc}_{\mathcal{C}}(\hat{v}_2) \in \{c_i, c_{i+1}\}$ and $h_{i,\bar{j}} \in [0, D(\mathcal{C})) \cup [d - D(\mathcal{C}), d)$.

We sketch the proof below, and refer the reader to the full version for details.

Proof sketch. We address each case individually.

1) If $(b_1, b_2, b_3) \in \{(1,1,0),(0,0,1)\}$ then we claim that $h_{i,\bar{j}} \geq n + D(\mathcal{C})$. Assume otherwise. If $h_{i,\bar{j}} \in [0, D(\mathcal{C}))$, then $g_j[D(\mathcal{C}) :] = w_i[D(\mathcal{C}) :]$, and $s_2(g_j) = s_3(g_j)$ are full chunks. Given the assumption that all the full chunks are decoded correctly, $b_2 = \mathrm{Maj}(s_2(x)) = \mathrm{Maj}(s_3(x)) = b_3$ but that contradicts our assumption for this case; so we conclude that $h_{i,\bar{j}} \notin [0, D(\mathcal{C}))$. Thus, $h_{i,\bar{j}} \in [D(\mathcal{C}), n + D(\mathcal{C}))$. But then then $s_1(g_j)$ and $s_2(g_j)$ are full chunks, and $s_1(g_j) \neq s_2(g_j)$. This implies that $b_1 \neq b_2$, again a contradiction. This shows $h_{i,\bar{j}} \geq n + D(\mathcal{C})$.
   Finally, the fact that $h_{i,\bar{j}} \geq n + D(\mathcal{C})$ implies that $\tilde{c}_1(g_j) = \tilde{c}_1(w_{i+1}) = c_{i+1}$, and $\tilde{c}_1(g_j)$ is a full chunk. Using the assumption of correct decoding of all full chunks, we see that $\mathrm{Enc}_{\mathcal{C}}(\hat{v}_1) = c_{i+1}$.
2) If $(b_1, b_2, b_3) \in \{(0,1,1),(1,0,0)\}$, then the conclusion follows by an argument nearly identical to Case 1.
3) If $(b_1, b_2, b_3) \in \{(0,0,0),(1,1,1)\}$, then we claim that $h_{i,\bar{j}} \in [0, D(\mathcal{C})) \cup [d - D(\mathcal{C}), d)$. Assume otherwise, then $s_1(g_j) = s_1(w_{i+1})$ and $s_3(g_j) = s_3(w_i)$ and they are full chunks. Now as i and $i+1$ do not have the same parity, $s_1(g_j) \neq s_3(g_j)$. As a result, if all full chunks are decoded correctly, we have that $b_1 \neq b_3$, which contradicts our assumption in this case. This proves our claim that $h_{i,\bar{j}} \in [0, D(\mathcal{C})) \cup [d - D(\mathcal{C}), d)$.
   If $h_{i,\bar{j}} \in [0, D(\mathcal{C}))$ then $\tilde{c}_1(g_j) = \tilde{c}_2(g_j) = c_i$; if $h_{i,\bar{j}} \in [d - D(\mathcal{C}), d)$ then $\tilde{c}_1(g_j) = \tilde{c}_2(g_j) = c_{i+1}$; and in either case both are full chunks. Using the assumption that all full chunks are decoded correctly, we see that $\mathrm{Enc}_{\mathcal{C}}(\hat{v}_1) = \mathrm{Enc}_{\mathcal{C}}(\hat{v}_2) \in \{c_i, c_{i+1}\}$, as desired.

□

A. Decoding Algorithm

Before we state our main algorithm (Algorithm 2 below), we include a helper algorithm, `compute-r` (Algorithm 1). This algorithm takes an index $i \in \{0, \ldots, 2^k - 1\}$ and returns r_i. Note that this is not trivial to do efficiently: If we wanted to compute r_i directly from the definition, that would require computing or storing $\Delta(w_\ell, w_{\ell+1})$ for all $\ell \leq i$ and adding them up, which may take time $\Omega(2^k)$. Instead, we do something much faster.

Algorithm 1 `compute-r`

Input: $i \in \{0, ..., 2^k - 1\}$
$\hat{r}_i = 0$
for $z \in \{0, \ldots, k-1\}$ **do**
  $\hat{r}_i = \hat{r}_i + 2\lfloor \frac{i+2^z}{2^{z+1}} \rfloor \cdot \|a_z\| + 3D(\mathcal{C})$ ▷ a_z is the z'th row of the generator matrix of $\mathcal{C}$.
end for
Return: $\hat{r}_i$

Lemma 2. The Algorithm `compute-r` (Algorithm 1) correctly computes r_i.

Proof. Recall that $r_i = \sum_{\ell=0}^{i-1} \Delta(w_\ell, w_{\ell+1})$. Consider a fixed difference $\Delta(w_\ell, w_{\ell+1})$. This is precisely

$$\Delta(w_\ell, w_{\ell+1}) = 2\|a_{z_\ell}\| + 3D(\mathcal{C}), \tag{3}$$

where z_ℓ is the unique index so that $\mathcal{R}_k(\ell)[z_\ell] \neq \mathcal{R}_k(\ell+1)[z_\ell]$: indeed, by Definition 1, $\Delta(c_\ell, c_{\ell+1}) = \|a_{z_\ell}\|$, and from that (3) follows from the definition of $\mathcal{W}$ (Definition 2). Thus, in order to compute

$$r_i = \sum_{\ell=0}^{i-1} (2\|a_{z_\ell}\| + 3D(\mathcal{C})),$$

it suffices to count how often each index $z \in \{0, \ldots, k-1\}$ shows up as some z_ℓ in that sum. This is precisely $\lfloor \frac{i+2^z}{2^{z+1}} \rfloor$, by the definition of $\mathcal{R}_k$. □

Our final algorithm is given in Algorithm 2. It is organized into the three cases of Lemma 1. To help the reader, we have included comments saying what each estimate "should" be. Here, "should" is under the assumption that each full chunk is decoded correctly.

Algorithm 2 $\mathrm{Dec}_{\mathcal{G}}$: Decoding algorithm for $\mathcal{G}$

Input: $x = g_j + \eta \in \mathbb{F}_2^d$
Output: $\hat{j} \in [N]$
for $\ell \in \{1, 2\}$ **do**
  $\hat{v}_\ell = \mathrm{Dec}_{\mathcal{C}}(\tilde{c}_\ell(x))$ ▷ Decode $\tilde{c}_1(x)$ and $\tilde{c}_2(x)$ individually to obtain $\hat{v}_1, \hat{v}_2 \in \{0, \ldots, 2^k - 1\}$.
end for
for $\ell \in \{1, 2, 3\}$ **do**
  $b_\ell = \mathrm{Maj}_{D(\mathcal{C})}(s_\ell(x))$ ▷ Decode each $s_\ell(x)$ to obtain $b_\ell \in \{0, 1\}$.
end for
  ▷ Below, in the comments we note what each value "should" be. This is what these values will be under the assumption that each full chunk is decoded correctly.
if $(b_1, b_2, b_3) \in \{(1,1,0),(0,0,1)\}$ **then**
  ▷ **Case 1:** $\mathrm{Enc}_{\mathcal{C}}(\hat{v}_1)$ should be c_{i+1}
  $\hat{\imath} = \mathcal{R}_k^{-1}(\hat{v}_1)$ ▷ $\hat{\imath}$ should be $i+1$
  $\hat{v} = \mathcal{R}_k(\hat{\imath} - 1)$ ▷ $\hat{v}$ should be $\mathcal{R}_k(i)$
  $\hat{c}_1 = \mathrm{Enc}_{\mathcal{C}}(\hat{v})$ ▷ $\hat{c}_1$ should be c_i
  $\hat{c}_2 = \mathrm{Enc}_{\mathcal{C}}(\hat{v}_1)$ ▷ $\hat{c}_2$ should be c_{i+1}
  $\hat{w}_1 = \mathrm{Enc}_{\mathcal{W}}(\hat{c}_1)$ ▷ $\hat{w}_1$ should be w_i
  $\hat{w}_2 = \mathrm{Enc}_{\mathcal{W}}(\hat{c}_2)$ ▷ $\hat{w}_2$ should be w_{i+1}
  $H' = \{\ell \in h_{\hat{\imath}-1} : \ell \geq n + D(\mathcal{C})\}$
  $u = \mathrm{Dec}_{\mathcal{U}}(x[H'] + \hat{w}_1[H'])$
    ▷ u is an estimate of $h_{i,\bar{j}} - \Delta(c_i, c_{i+1}) - D(\mathcal{C})$
  $\hat{j} = u + D(\mathcal{C}) + \Delta(\hat{c}_1, \hat{c}_2) + \texttt{compute-r}(\hat{\imath} - 1)$
else if $(b_1, b_2, b_3) \in \{(0,1,1),(1,0,0)\}$ **then**
  ▷ **Case 2:** $\mathrm{Enc}_{\mathcal{C}}(\hat{v}_2)$ should be c_i
  $\hat{\imath} = \mathcal{R}_k^{-1}(\hat{v}_2)$ ▷ $\hat{\imath}$ should be i
  $\hat{v} = \mathcal{R}_k(\hat{\imath} + 1)$ ▷ $\hat{v}$ should be $\mathcal{R}_k(i+1)$
  $\hat{c}_1 = \mathrm{Enc}_{\mathcal{C}}(\hat{v}_2)$ ▷ $\hat{c}_1$ should be c_i
  $\hat{c}_2 = \mathrm{Enc}_{\mathcal{C}}(\hat{v})$ ▷ $\hat{c}_2$ should be c_{i+1}

$\hat{w}_1 = \mathrm{Enc}_{\mathcal{W}}(\hat{c}_1)$ ▷ $\hat{w}_1$ should be w_i

$\hat{w}_2 = \mathrm{Enc}_{\mathcal{W}}(\hat{c}_2)$ ▷ $\hat{w}_2$ should be w_{i+1}

$H' = \{\ell \in h_{\hat{\imath}-1} : \ell < n + 2D(\mathcal{C})\}$

$u = \mathrm{Dec}_{\mathcal{U}}(x[H'] + \hat{w}_1[H'])$

▷ u is an estimate of $h_{i,\hat{j}} \le 2D(\mathcal{C}) + n$

$\hat{j} = u + \texttt{compute-r}(\hat{\imath})$

else if $(b_1, b_2, b_3) \in \{(0,0,0), (1,1,1)\}$ **then**

▷ **Case 3**: $\mathrm{Enc}_{\mathcal{C}}(\hat{v}_1)$ and $\mathrm{Enc}_{\mathcal{C}}(\hat{v}_2)$ should be equal to each other, and to either c_i or c_{i+1}, but we need to figure out which one.

$\hat{\imath} = \mathcal{R}_k^{-1}(\hat{v}_1)$ ▷ $\hat{\imath}$ should be either i or $i+1$

$\hat{v} = \mathcal{R}_k(\hat{\imath} - 1)$ ▷ $\hat{v}$ should be $\mathcal{R}_k(i)$ or $\mathcal{R}_k(i-1)$

$\hat{c}_1 = \mathrm{Enc}_{\mathcal{C}}(\hat{v})$ ▷ $\hat{c}_1 = c_i$ if $\hat{\imath} = i+1$; c_{i-1} if $\hat{\imath} = i$

$\hat{c}_2 = \mathrm{Enc}_{\mathcal{C}}(\hat{v}_1)$ ▷ $\hat{c}_2 = c_{i+1}$ if $\hat{\imath} = i+1$; c_i if $\hat{\imath} = i$

$\hat{w} = \mathrm{Enc}_{W}(\hat{c}_2)$ ▷ $\hat{w} = w_{i+1}$ if $\hat{\imath} = i+1$; w_i if $\hat{\imath} = i$

$u_1 = \mathrm{Dec}_{\mathcal{U}}(x[< D(\mathcal{C})] + b_1^{D(\mathcal{C})})$

▷ u_1 is an estimate of $h_{i,\hat{j}} < D(C)$ if $\hat{w} = w_i$

$u_2 = \mathrm{Dec}_{\mathcal{U}}(x[> 2D(\mathcal{C}) + 2n] + \bar{b}_1^{D(\mathcal{C})})$

▷ u_2 is an estimate of $h_{i,\hat{j}} - 2D(\mathcal{C}) - 2\Delta(c_i, c_{i+1})$ if $\hat{w} = w_{i+1}$

$\hat{j}_1 = u_1 + \texttt{compute-r}(\hat{\imath})$ ▷ Estimate j if $\hat{w} = w_i$

$\hat{j}_2 = u_2 + 2D(\mathcal{C}) + 2\Delta(\hat{c}_1, \hat{c}_2) + \texttt{compute-r}(\hat{\imath} - 1))$

▷ Estimate j if $\hat{w} = w_{i+1}$

$\hat{g}_1 = \mathrm{Enc}_{\mathcal{G}}(\hat{j}_1)$

$\hat{g}_2 = \mathrm{Enc}_{\mathcal{G}}(\hat{j}_2)$

$\hat{j} = \min_{j' \in \{\hat{j}_1, \hat{j}_2\}} \Delta(x, \hat{g}_{j'})$

end if

B. Analysis

Next, we provide formal statements about the correctness and running time of Algorithm 2. Due to space constraints, we state only the main results and defer all proofs to the full version, and only offer intuition in this extended abstract.

We begin with the running time, which follows immediately from inspection of Algorithms 1 and 2.

Lemma 3. Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a code with rate $\Omega(1)$. Suppose that $\mathrm{Dec}_{\mathcal{C}}$ runs in time $T_{\mathrm{Dec}_{\mathcal{C}}}(n)$, $\mathrm{Enc}_{\mathcal{C}}$ runs in time $T_{\mathrm{Enc}_{\mathcal{C}}}(n)$, and $D(\mathcal{C}) = o(n)$. Let $A_{\mathcal{C}}$ be the generator matrix of $\mathcal{C}$, with rows a_z for $z \in \{0, \ldots, 2^k - 1\}$. Suppose that $\|a_z\|$ can be computed in time $O(1)$. Then the running time is of $\mathrm{Dec}_{\mathcal{G}}$ is $O(T_{\mathrm{Dec}_{\mathcal{C}}}(d) + T_{\mathrm{Enc}_{\mathcal{C}}}(d) + d)$, and the running time of $\mathrm{Enc}_{\mathcal{G}}$ is $O(T_{\mathrm{Enc}_{\mathcal{C}}}(d))$.

Remark 1 (Time to compute $\|a_z\|$)**.** If $\mathcal{C}$ is, say, a Reed-Muller code $\mathcal{RM}(r, m)$, then indeed, given z, $\|a_z\|$ can be computed in time $O(1)$: if the binary expansion of z has weight $t \le r$, then the corresponding row has weight 2^{m-t}. For codes that may not have closed-form expressions for their generator matrices, we can pre-compute each $\|a_z\|$ (in total time $O(d^2)$) and store them to allow for $O(1)$ lookup time.[3]

Finally, we state our main correctness theorem.

[3]If a lookup table is not desirable and the $\|a_z\|$ cannot otherwise be computed on the fly, then our algorithm still works, and $\mathrm{Dec}_{\mathcal{G}}$ runs in time at most $O(T_{\mathrm{Dec}_{\mathcal{C}}}(d) + T_{\mathrm{Enc}_{\mathcal{C}}}(d) + d^2)$, where we recall that $d = \Theta(n)$ and $O(\log N)$.

Theorem 1. Fix $p \in (0, 1/2)$. Let $\mathcal{C} \subseteq \mathbb{F}_2^n$ be a linear code. Let $\mathcal{G}$ be defined as in Definition 5 from $\mathcal{C}$. Let $\eta_p \in \mathbb{F}_2^d \sim \mathrm{Ber}(p)^d$ Then

$$\Pr\left[|j - \mathrm{Dec}_{\mathcal{G}}(\mathrm{Enc}_{\mathcal{G}}(j) + \eta_p)| > t\right] \le \gamma \exp(-\alpha t) + 5P_{\text{fail}}(\mathcal{C})$$

where α and γ are constants given by $\alpha = -\frac{(1-2p)^2}{4p+2}$ and $\gamma = \frac{2}{1-\exp(-\alpha)}$.

Unfortunately, we do not have space for the proof (see the full version), but we give a quick sketch here.

First, we argue that, under the assumption that all full chunks are decoded correctly, that the values computed in Algorithm 1 (for example, $\hat{\imath}, \hat{w}_1, \hat{w}_2, \hat{c}_1, \hat{c}_2$, and so on) are what they "should" be. That is, we argue that these are the values that the comments in Algorithm 2 indicate.

Second, we show that, again under the assumption that all full chunks are decoded correctly, we always have

$$|j - \hat{j}| = \Delta(g_j, g_{\hat{j}}), \tag{4}$$

where $\hat{j}$ is the estimate of j returned by $\mathrm{Dec}_{\mathcal{G}}$. This involves some case analysis and stepping carefully through Algorithm 2.

Next, let $\mathcal{S}$ be the bad event that some full chunk is decoded incorrectly. We write

$$\begin{aligned}
&\Pr[|j - \hat{j}| > t] \\
&\quad = \Pr\left[|j - \hat{j}| > t \,|\, \bar{\mathcal{S}}\right] \cdot \Pr[\bar{\mathcal{S}}] + \Pr\left[|j - \hat{j}| > t \,|\, \mathcal{S}\right] \cdot \Pr[\mathcal{S}] \\
&\quad \le \Pr\left[|j - \hat{j}| > t \,|\, \bar{\mathcal{S}}\right] + \Pr[\mathcal{S}],
\end{aligned}$$

and bound each term separately.

To bound the first term, we show that if the algorithm returns $\hat{j}$ rather than the correct answer j, then it was because $g_{\hat{j}}$ was closer to the received word x than g_j was, that is, $\Delta(g_{\hat{j}}, x) \le \Delta(g_j, x)$. As a result, this can only happen if the noise vector η_p has weight at least $\Delta(g_j, g_{\hat{j}})$, which by a Chernoff bound happens with probability at most $\exp(-\alpha\Delta(g_j, g_{\hat{j}})$ for some constant α. Using (4), this implies that the probability of returning $\hat{j}$ is at most $\exp(-\alpha|j - \hat{j}|)$. Finally, we can conclude that

$$\Pr\left[|j - \hat{j}| \ge t \,|\, \bar{\mathcal{S}}\right] \le \sum_{z=t}^{\infty} 2\exp(-\alpha z) = \frac{2\exp(-\alpha t)}{1 - \exp(-\alpha)},$$

which bounds the first term.

Finally, the second term $\Pr[\mathcal{S}]$ can be bounded by $O(P_{\text{fail}}(\mathcal{C}))$. This is because if any full chunk is decoded incorrectly, then either $\mathcal{C}$'s decoder failed, or else the Maj decoder failed on a string of length $\Delta(\mathcal{C})$, which we show happens with probability at most $P_{\text{fail}}(\mathcal{C})$ as well.

Altogether, this line of reasoning establishes Theorem 1. We refer the reader to the full version for more details.

ACKNOWLEDGMENT

MW and DF are partially supported by NSF Grants CCF-2231157 and CCF-2133154. The first author thanks Rasmus Pagh for bringing our attention to this problem.

REFERENCES

[1] D. Fathollahi and M. Wootters, "Improved construction of robust gray codes," 2024, arXiv eprint 2401.15291. [Online]. Available: https://arxiv.org/abs/2401.15291

[2] D. R. Lolck and R. 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)*. SIAM, 2024, pp. 1050–1066.

[3] M. Aumüller, C. J. Lebeda, and R. 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*, 2021, pp. 1223–1236.

[4] J. Acharya, Y. Liu, and Z. Sun, "Discrete distribution estimation under user-level local differential privacy," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, pp. 8561–8585.

[5] J. Acharya, C. Canonne, Y. Liu, Z. Sun, and H. Tyagi, "Distributed estimation with multiple samples per user: Sharp rates and phase transition," *Advances in neural information processing systems*, vol. 34, pp. 18 920–18 931, 2021.

[6] L. Xiao, X.-G. Xia, and Y.-P. Wang, "Exact and robust reconstructions of integer vectors based on multidimensional chinese remainder theorem (md-crt)," *IEEE Transactions on Signal Processing*, vol. 68, pp. 5349–5364, 2020.

[7] W. Wang and X.-G. Xia, "A closed-form robust chinese remainder theorem and its performance analysis," *IEEE Transactions on Signal Processing*, vol. 58, no. 11, pp. 5655–5666, 2010.

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

[9] E. Arikan, "A performance comparison of polar codes and reed-muller codes," *IEEE Communications Letters*, vol. 12, no. 6, pp. 447–449, 2008.

[10] V. Guruswami and P. Xia, "Polar codes: Speed of polarization and polynomial gap to capacity," *IEEE Transactions on Information Theory*, vol. 61, no. 1, pp. 3–16, 2014.

[11] M. Mondelli, S. H. Hassani, and R. L. Urbanke, "Unified scaling of polar codes: Error exponent, scaling exponent, moderate deviations, and error floors," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 6698–6712, 2016.

[12] H.-P. Wang, T.-C. Lin, A. Vardy, and R. Gabrys, "Sub-4.7 scaling exponent of polar codes," *IEEE Transactions on Information Theory*, 2023.

[13] F. Gray, "Pulse code communication," Mar. 17 1953, uS Patent 2,632,058.

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