# Capacity-Achieving Gray Codes

**Venkatesan Guruswami**
University of California, Berkeley, CA, USA

**Hsin-Po Wang**
University of California, Berkeley, CA, USA

**Abstract**

To ensure differential privacy, one can reveal an integer fuzzily in two ways: (a) add some Laplace noise to the integer, or (b) encode the integer as a binary string and add iid BSC noise. The former is simple and natural while the latter is flexible and affordable, especially when one wants to reveal a sparse vector of integers. In this paper, we propose an implementation of (b) that achieves the capacity of the BSC with positive error exponents. Our implementation adds error-correcting functionality to Gray codes by mimicking how software updates back up the files that are getting updated ("coded Gray code"). In contrast, the old implementation of (b) interpolates between codewords of a black-box error-correcting code ("Grayed code").

**2012 ACM Subject Classification** Mathematics of computing → Coding theory; Security and privacy → Privacy-preserving protocols

**Keywords and phrases** Gray codes, capacity-achieving codes, differential privacy

**Digital Object Identifier** 10.4230/LIPIcs.APPROX/RANDOM.2024.65

**Category** RANDOM

**Funding** *Venkatesan Guruswami*: NSF grant CCF-2210823 and a Simons Investigator Award

## 1 Introduction

Differential privacy is the art of publishing collective facts without leaking any detail of any user. A mathematically rigorous way to do so is adding noise to an aggregation function that is Lipschitz continuous (sometimes of bounded variation) in every argument. More concretely, suppose that we are interested in a feature $\varphi\colon \{0,1\}^n \to [m]$ that satisfies

$$|\varphi(u) - \varphi(u')| \leqslant 1, \text{ for } u \coloneqq (u_1,\dots,u_i,\dots,u_n) \text{ and } u' \coloneqq (u_1,\dots,1{-}u_i,\dots,u_n),$$

i.e., changing the data of the $i$th user does not change the feature too much. Then publishing $\varphi(u) + L$, where $L$ follows the Laplace distribution with decay rate $\varepsilon$, is $\varepsilon$-differentially private [3]. That is,

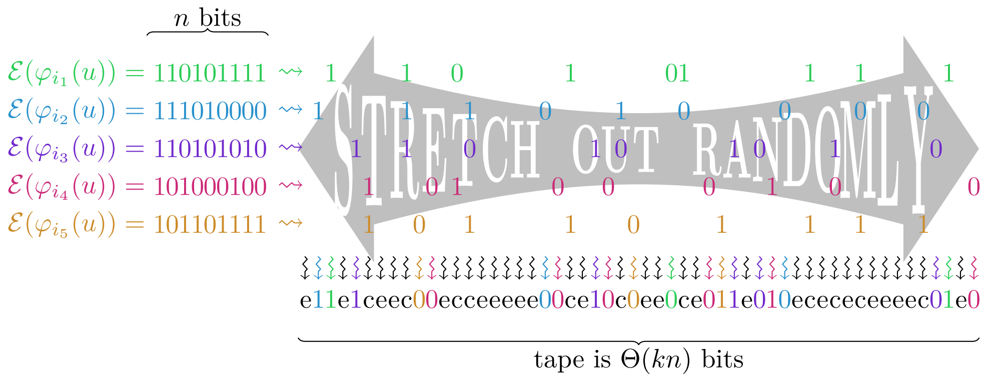
$$\operatorname{Prob}\{\varphi(u) + L < t\} \leqslant \exp(\varepsilon) \operatorname{Prob}\{\varphi(u') + L < t\} \tag{1}$$

for any number $t \in \mathbb{R}$, meaning that a data broker will have a hard time telling if $u_i$ is 0 or 1.

Publishing $\varphi(u) + L$ is called the *Laplace mechanism* [3]. It is optimal[1] privacy-wise as (1) assumes equality half of the time. But it turns out to be randomness-costly and space-inefficient when we have many features $\varphi_1,\dots,\varphi_\ell$ to publish, wherein only $k \ll \ell$ of them are non-zero[2] for a given $x$. In this case, the Laplace mechanism will add noise to

[1] Note that we can always choose to publish $\varphi(u) + 100L$, which is more private than $\varphi(u) + L$ by being less informative and less useful. We say that $\varphi(u) + L$ is optimal because it strikes a balance between (1) and utility.

[2] For example, $\varphi_i(u)$ could be the number of times the $i$th English word was mentioned in a forum archive $u$. Most word counts are going to be zero.



$n$ bits
$\mathcal{E}(\varphi_{i_1}(u)) = 110101111$
$\mathcal{E}(\varphi_{i_2}(u)) = 111010000$
$\mathcal{E}(\varphi_{i_3}(u)) = 110101010$
$\mathcal{E}(\varphi_{i_4}(u)) = 101000100$
$\mathcal{E}(\varphi_{i_5}(u)) = 101101111$
STRETCH OUT RANDOMLY
e11e1ceec00ecceeeee00ce10c0ee0ce011e010ecececeeeec01e0
tape is $\Theta(kn)$ bits

**Figure 1** A space-efficient differential privacy mechanism. Step 1: encode integers as binary strings. Step 2: spread out the bits. Step 3: superimpose them on a tape. Features $\varphi_{i_1}(u), \ldots, \varphi_{i_5}(u)$ are the ones that are nonzero. Labels c and e mean collision and empty, respectively; collisions will be replaced by random bits; empty places will be filled with 0.

all $\varphi_i(u)$ and then publish all $\ell$ of them. For one, this means that we are forced to sample Laplace distribution $\ell$ times. Even if we can afford that, the output will be $\Omega(\ell \log m)$ in size ($m$ is an upper bound on the $\varphi$'s) while the raw data is only $\mathcal{O}(k \log(\ell) \log(m))$.

A brilliant idea of Lolck and Pagh [6], which is a generalization of an earlier work by Aumüller, Lebeda, and Pagh [1], reduces the space requirement as well as the sampling cost. The idea is that, instead of working on the ordered field $\mathbb{R}$, we encode each $\varphi_i(u)$ as a binary string $\mathcal{E}(\varphi_i(u)) \in \{0,1\}^{1\times n}$ and put the bits of $\mathcal{E}(\varphi_i(u))$ at $n$ random places on a tape of length $\Theta(kn)$. This is illustrated in Figure 1. Note that $\mathcal{E}(\varphi_{i_1}(u))$ and $\mathcal{E}(\varphi_{i_2}(u))$ might end up choosing the same random places. Such a collision is resolved, fairly, by putting a random bit there. These random bits together with additional random bit-flips will play the role of the Laplace noise – protecting privacy by making precise decoding impossible.

One problem remains: To what extent can we translate the binary tape back to real numbers? This motivates the definition of robust Gray codes.

## 1.1 Robust Gray Codes

A Gray code encodes integers as binary strings such that any two consecutive strings differ at exactly one place. A popular construction of Gray codes is via the ruler sequence [7, A001511]

$$\rho_j \coloneqq \text{the greatest number } r \text{ such that } 2^r \text{ divides } 2j. \tag{2}$$

The first few terms read $1, 2, 1, 3, 1, 2, 1, 4, 1, 2, 1, 3, 1, 2, 1, 5$. Then, the $(j+1)$th string of the $k$-bit reflected Gray code is obtained by flipping the $\min(\rho_j, k)$th bit of the $j$th string. For simplicity, we will write $\min(\rho_j, k)$ as $\rho_j$, and so we can write $g^{j+1} = g^j + e^{\rho_j}$ instead of $g^j + e^{\min(\rho_j,k)}$, where $e^r$ is the $r$th standard basis vector of length $k$. As an example, when $k = 4$,

$$\begin{array}{ll}
g^1 = 0\,0\,0\,0 & \\
g^2 = 1\,0\,0\,0 & \rho_1 = 1 \\
g^3 = 1\,1\,0\,0 & \rho_2 = 2 \\
g^4 = 0\,1\,0\,0 & \rho_3 = 1 \\
g^5 = 0\,1\,1\,0 & \rho_4 = 3 \\
g^6 = 1\,1\,1\,0 & \rho_5 = 1 \\
g^7 = 1\,0\,1\,0 & \rho_6 = 2 \\
g^8 = 0\,0\,1\,0 & \rho_7 = 1 \\
g^9 = 0\,0\,1\,1 & \rho_8 = 4
\end{array}$$

are the first nine strings. (Digits that are flipped are highlighted.)

A *robust Gray code* [1, 6] encodes integers as binary strings such that they can be fuzzily recovered even if some bits are erased or corrupted. Given the motivational Figure 1, let us use the binary symmetric channels (BSC) with crossover probability $p \in (0, 1/2)$ to model the errors. Then a robust Gray code is a pair of encoder

$$\mathcal{E}\colon [m] \to \{0,1\}^{1\times n}$$

and decoder

$$\mathcal{D}\colon \{0,1\}^{1\times n} \to [m]$$

such that (a) $\mathcal{E}(x)$ and $\mathcal{E}(x+1)$ differ by one bit and (b)

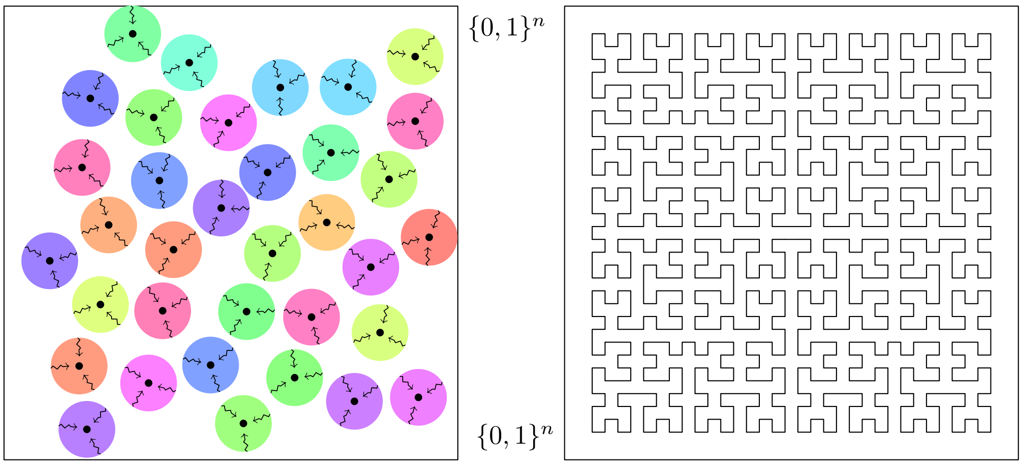
$$\operatorname{Prob}\Big\{\Big|\mathcal{D}\big(\mathrm{BSC}_p^n(\mathcal{E}(x))\big) - x\Big| > t\Big\} < 2^{-\Omega(n)} + 2^{-\Omega(t)} \tag{3}$$

for all $x \in [m-1]$ and all $t > 1$. Here, $\mathrm{BSC}_p^n$ flips each of the $n$ bits with probability $p$. Note that (3) is almost as good as the Laplace mechanism in that $2^{-\Omega(t)}$ decays exponentially in $t$. The only catch is that when $t \gg n$, the other error term $2^{-\Omega(n)}$ dominates $2^{-\Omega(t)}$. This $2^{-\Omega(n)}$ is unavoidable because there is always[3] a $2^{-\mathcal{O}(n)}$ chance that BSC will flip all ones to zero.

Apart from robustness, we also care about space efficiency. We know that, by Shannon's theory, the code rate $\log_2(m)/n$ cannot exceed the capacity of $\mathrm{BSC}_p$, which is $1 + p\log_2(p) - (1-p)\log_2(1-p)$. But how close can they be? Before our work, Lolck and Pagh's construction [6] achieves $1/4$ of the capacity ($1/3$ in [6, Appendix A]) and Fathollahi and Wootters's construction [4] achieves $1/2$ of the capacity. This means that the latter uses half of the space to achieve the same privacy level.

In this work, and in a concurrent work by Con, Fathollahi, Gabrys, and Yaakobi [2], we will show that the capacity can be achieved. This means that, subject to the framework of Figure 1, the tradeoff between privacy and space is now asymptotically tight. We also show that our code has linear encoding and decoding complexity, meaning that even the speed cannot be significantly improved.

[3] Note that we implicitly assume that $p$ is bounded away from zero. This is a common practice in coding theory where channel parameters, $p$ in this case, are fixed while the other parameters vary. Also note that $\mathrm{BSC}_p$ here plays the role of the Laplace noise, so it would make less sense to have $p$ too close to zero unless, of course, one is aiming for some special privacy regime.



**(a)** An error-correcting code is some points that can be decoded up to some radius.

**(b)** A gray code is a Hamiltonian cycle that only goes in the cardinal directions.

**Figure 2** Figurative illustrations of error-correcting codes and Gray codes.

## 1.2 Previous approaches

Earlier works [6, 4] baked robust Gray codes with the following recipe.

- Take a good $[n, k]$-error correcting code $\mathcal{C} = \{c^1, c^2, \dots, c^{2^k}\} \subset \{0,1\}^{1\times n}$.
- Let $\mathcal{E}$ map "milestone" integers $1 =: \mu_1 < \mu_2 < \cdots < \mu_{2^k} := m$ to the codewords of $\mathcal{C}$, i.e., $\mathcal{E}(\mu_j) := c^j$.
- "Interpolate" between the milestones. That is, if $x \in [\mu_j, \mu_{j+1}]$, then the prefix of $\mathcal{E}(x)$ will come from $\mathcal{E}(\mu_j)$ and the suffix from $\mathcal{E}(\mu_{j+1})$.

The technicality is with the third bullet point. A decoder of $\mathcal{C}$ can translate $\mathcal{E}(x)$ back to $\mu_j$ if $x$ is close enough to $\mu_j$. But there is going to be a middle ground between $\mu_j$ and $\mu_{j+1}$ such that the decoder will be confused.

To eliminate the confusion, Lolck and Pagh [6] proposed the following data structure

$$\mathcal{E}(\mu_j) := c^j \| c^j \| c^j \| c^j \in \{0,1\}^{1\times 4n},$$

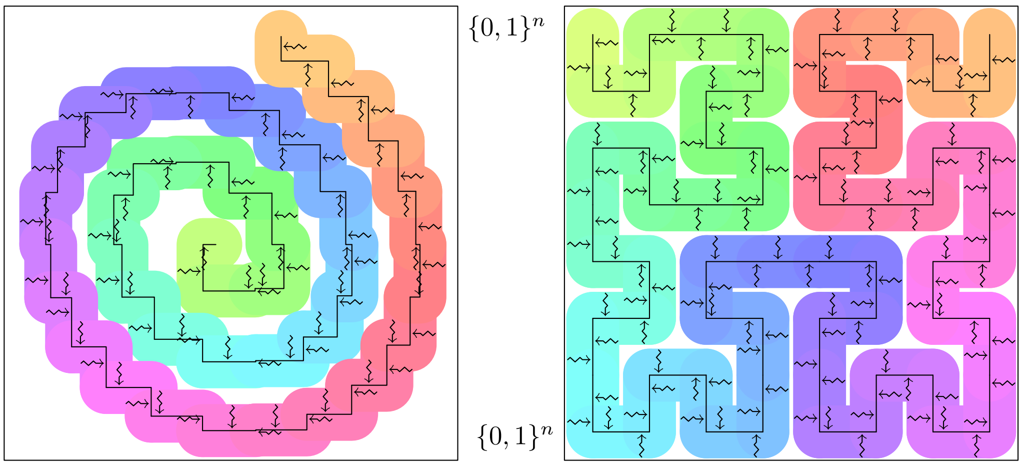
where $\|$ is the string concatenation operator. They then interpolate between consecutive milestones $\mu_j$ and $\mu_{j+1}$ as

$$\begin{aligned}
\mathcal{E}(\mu_{j.1}) &:= c^j \,\|\, c^j \,\|\, c^j \,\|\, c^j, \\
\mathcal{E}(\mu_{j.2}) &:= c^{j+1} \| c^j \,\|\, c^j \,\|\, c^j, \\
\mathcal{E}(\mu_{j.3}) &:= c^{j+1} \| c^{j+1} \| c^j \,\|\, c^j, \\
\mathcal{E}(\mu_{j.4}) &:= c^{j+1} \| c^{j+1} \| c^{j+1} \| c^j, \\
\mathcal{E}(\mu_{j.5}) &:= c^{j+1} \| c^{j+1} \| c^{j+1} \| c^{j+1}
\end{aligned}$$

for some minor milestones $\mu_j =: \mu_{j.1} < \mu_{j.2} < \mu_{j.3} < \mu_{j.4} < \mu_{j.5} := \mu_{j+1}$. Note that only one copy is undergoing interpolation at any given time (which is highlighted). So the advantage of repeating $c^j$ four times is that there are always two other copies that will decode to the same codeword. To elaborate, between $\mu_{j.1}$ and $\mu_{j.3}$, the two $c^j$ to the right will decode correctly; between $\mu_{j.3}$ and $\mu_{j.5}$, the two $c^{j+1}$ to the left will decode correctly.

Later, Fathollahi and Wootters [4] streamlined the data structure from $4n$ bits to $(2+3\varepsilon)n$ bits by using *buffers* – consecutive zeros and ones. They map milestones to

$$\mathcal{E}(\mu_j) := 0^{\varepsilon n} \| c^j \| 0^{\varepsilon n} \| c^j \| 0^{\varepsilon n} \in \{0,1\}^{1\times(2+3\varepsilon)n}$$



**(a)** "Grayed code": Old approach takes an error correcting code and then interpolates between codewords.

**(b)** "Coded Gray code": New approach multiplies a Gray code with the generator matrix of an error-correcting code.

**Figure 3** Old approach (not capacity-achieving) versus new approach (capacity-achieving).

if $j$ is even, and to

$$\mathcal{E}(\mu_j) \coloneqq 1^{\varepsilon n} \| c^j \| 1^{\varepsilon n} \| c^j \| 1^{\varepsilon n} \in \{0,1\}^{1\times(2+3\varepsilon)n}$$

if $j$ is odd. They then interpolate between the milestones as

$$\begin{aligned}
\mathcal{E}(\mu_{j.1}) &\coloneqq 0^{\varepsilon n} \| \; c^j \; \| 0^{\varepsilon n} \| \; c^j \; \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_{j.2}) &\coloneqq 1^{\varepsilon n} \| \; c^j \; \| 0^{\varepsilon n} \| \; c^j \; \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_{j.3}) &\coloneqq 1^{\varepsilon n} \| c^{j+1} \| 0^{\varepsilon n} \| \; c^j \; \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_{j.4}) &\coloneqq 1^{\varepsilon n} \| c^{j+1} \| 1^{\varepsilon n} \| \; c^j \; \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_{j.5}) &\coloneqq 1^{\varepsilon n} \| c^{j+1} \| 1^{\varepsilon n} \| c^{j+1} \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_{j.6}) &\coloneqq 1^{\varepsilon n} \| c^{j+1} \| 1^{\varepsilon n} \| c^{j+1} \| 1^{\varepsilon n}
\end{aligned}$$

for some minor milestones $\mu_j \eqqcolon \mu_{j.1} < \mu_{j.2} < \mu_{j.3} < \mu_{j.4} < \mu_{j.5} < \mu_{j.6} \coloneqq \mu_{j+1}$. In this construction, the decoder is left with two, not four, copies of $c^j$. It knows that the one sandwiched between $0^{\varepsilon n}$ and $1^{\varepsilon n}$ is the one undergoing interpolation, and hence the other one will decode correctly. To be more precise, between $\mu_{j.1}$ and $\mu_{j.4}$, the left one is undergoing interpolation and the right $c^j$ is trustworthy; between $\mu_{j.3}$ and $\mu_{j.6}$, the right one is undergoing interpolation and the left $c^{j+1}$ is trustworthy.

## 1.3 New approach

*While this paper was in preparation, it came to our attention that Con, Fathollahi, Gabrys, Wootters, and Yaakobi have achieved similar results, but with different techniques [2]. In particular, their approach uses code concatenation.*

In this and the concurrent work by Con, Fathollahi, Gabrys, Wootters, and Yaakobi, we aim to rightsize the length to $n + \Theta(\varepsilon n)$ bits. While their work uses code concatenation, we begin with a generator matrix $A \in \{0,1\}^{k\times n}$ of some error-correcting code. We then reorder the codewords $c^1, \dots, c^{2^k}$ using Gray code:

$$c^{j+1} = g^{j+1}A = (g^j + e^{\rho_j})A = c^j + A^{\rho_j} \in \{0,1\}^{1\times n}.$$

Here, $g^j$ is the $j$th string of the Gray code, $e^{\rho_j}$ is the $\rho_j$th cardinal vector, and $A^{\rho_j}$ is the $\rho_j$th row of $A$, all as row vectors. Our data structure will look like

$$c^j \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n}$$

or

$$c^j \| 1^{\varepsilon n} \| \rho_j \| \beta^j \| 1^{\varepsilon n} \| \rho_j \| \beta^j \| 1^{\varepsilon n}$$

depending on the parity of $j$, Here, $\beta^j$ is a subvector of $c^j$ obtained by collecting bits where $A^{\rho_j}$ has 1. More precisely, if $A^{\rho_j}$ has 1 at indices $i_1, i_2, \ldots, i_w$, then $\beta^j := c^j_{i_1} c^j_{i_2} \cdots c^j_{i_w} \in \{0,1\}^{1\times w}$, where $w$ is the Hamming weight of $A^{\rho_j}$.

The purpose of keeping $\rho_j$ in $\mathcal{E}$ is to take note of which row of $A$ we are going to add to $c^j$ to obtain $c^{j+1}$. The purpose of keeping $\beta^j$ in $\mathcal{E}$ is to back up the bits of $c^j$ that are going to be modified. We then interpolate between minor milestones

$$\begin{array}{l}
c^j \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n}, \\
c^{j+1} \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n}, \\
c^{j+1} \| 1^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n}, \\
c^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^j \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n}, \\
c^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^{j+1} \| 0^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n}, \\
c^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^{j+1} \| 1^{\varepsilon n} \| \rho_j \| \beta^j \| 0^{\varepsilon n}, \\
c^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^j \| 0^{\varepsilon n}, \\
c^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^{j+1} \| 0^{\varepsilon n}, \\
c^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^{j+1} \| 1^{\varepsilon n} \| \rho_{j+1} \| \beta^{j+1} \| 1^{\varepsilon n},
\end{array}$$

Note that we use Fathollahi and Wootters's data structure to protect $\rho_j$ and $\beta^j$, and so the backup data is almost always available.[4]

An analogy of this construction is to think of $c_j$ as the state of our computer at version $j$. Now a software update comes in and attempts to add $A^{\rho_j}$ to $c_j$. To avoid messing things up, the updater backs up the files that are going to be updated, which are at $i_1, \ldots, i_w$; and $\beta^j$ is the backup data.

Our construction, at any code rate below capacity, achieves positive error exponents and linear encoding and decoding complexity.

▶ **Theorem 1** (Main theorem). *Fix a BSC with $p \in (0, 1/2)$ and a gap to capacity $\varepsilon > 0$. For sufficiently large $n$, there exists a pair of encoder $\mathcal{E}\colon [m] \to \{0,1\}^{1\times n}$ and decoder $\mathcal{D}\colon \{0,1\}^{1\times n} \to [m]$ with code size $m > 2^{(\mathrm{Capacity}(\mathrm{BSC}_p)-\varepsilon)n}$ such that (a) $\mathcal{E}(x)$ and $\mathcal{E}(x+1)$ differ by one bit and (b)*

$$\operatorname{Prob}\Big\{\Big|\mathcal{D}\big(\mathrm{BSC}_p^n(\mathcal{E}(x))\big) - x\Big| > t\Big\} < 2^{-\Omega(n)} + 2^{-\Omega(t)}$$

*for all $x \in [m-1]$ and all $t > 1$. Moreover, the time complexity of $\mathcal{E}$ and $\mathcal{D}$ scales[5] linearly in $n$.*

## Organization

The rest of the paper is dedicated to proving Theorem 1.

[4] There is no reason not to protect $\rho_j$ and $\beta^j$ using error-correcting codes. We omit that here but will discuss in the formal proof.
[5] exponentially in $1/\operatorname{poly}(\varepsilon)$

## 2 Proof of Theorem 1

Fix a crossover probability $p$ and a gap to capacity $\varepsilon > 0$. Let $n$ and $k$ be very large.

### 2.1 The building blocks $\mathcal{B}$ and $\mathcal{C}$

We begin with a linear code $\mathcal{B}$ with block length $\varepsilon n$ and dimension $\varepsilon k$. Using well-known constructions [5, Theorem 8], we can make the code rate $k/n$ $\varepsilon$-close to the capacity if $n$ is large enough. Moreover, the encoding and decoding complexity can be made linear in $n$. Let $B$ be the generator matrix of $\mathcal{B}$.

We stack $B$ to construct a larger generator matrix

$$A := \begin{bmatrix} B & \bar{B} & & & & \\ & B & \bar{B} & & & \\ & & B & \bar{B} & & \\ & & & \ddots & \ddots & \\ & & & & B & \bar{B} \end{bmatrix} \in \{0,1\}^{k \times (1+\varepsilon)n} \tag{4}$$

and denote the corresponding code by $\mathcal{C} \subset \{0,1\}^{1\times(1+\varepsilon)n}$. Here, $\bar{B}$ is the bitwise complement of $B$. We put $\bar{B}$ next to $B$ so that all rows of $[B \quad \bar{B}]$ has the same Hamming weight, $\varepsilon n$. This means that the backup data $\beta^j$ will be exactly $\varepsilon n$ bits long. We repeat $B$ and $\bar{B}$ $1/\varepsilon$ times so that $A$ has block length $(1+\varepsilon)n$ and dimension $k$. This makes the code rate of $\mathcal{C}$ $2\varepsilon$-close to the capacity.

Decoding $\mathcal{C}$ is straightforward. Given a received word $y \in \{0,1\}^{1\times(1+\varepsilon)n}$, apply $\mathcal{B}$'s decoder to $y_1, \ldots, y_{\varepsilon n}$ to obtain $x_1, \ldots, x_{\varepsilon n}$. Subtract the influence of $x_1, \ldots, x_{\varepsilon n}$ from $y_{\varepsilon n+1}, \ldots, y_{2\varepsilon n}$ and apply $\mathcal{B}$'s decoder to obtain $x_{\varepsilon n+1}, \ldots, x_{2\varepsilon n}$. Repeat this process until we obtain $x_n$.

We also use $\mathcal{B}$ to protect the row index $\rho_j \in [k]$ and the backup data $\beta^j \in \{0,1\}^{\varepsilon n}$. Denote by $\mathcal{B}(\rho_j, \beta^j)$ the result of encoding these $\log_2(k) + \varepsilon n$ bits of information using

$$\frac{2n}{k} \geqslant \left\lceil \frac{\log_2(k) + \varepsilon n}{\varepsilon k} \right\rceil$$

blocks of $\mathcal{B}$. This means that $\mathcal{B}(\rho_j, \beta^j)$ has length $2\varepsilon n^2/k$.

### 2.2 Encoding $\mathcal{E}$

Recall that $\rho_j$ is the ruler sequence defined in (2) capped at $k$. Recall that $g^j$ is the $j$th string of the Gray code and is obtained by flipping the $\min(\rho_{j-1}, k)$th bit of $g^{j-1}$. We assume an ordering on the codewords $\mathcal{C} = \{c^1, \ldots, c^{2^k}\}$ by Gray code, i.e., $c^j := g^j A$. Let $\beta^j$ be the subvector of $c^j$ obtained by deleting the bits where $A^{\rho_j}$ has 0.

We now place the milestones at

$$\mu_j := j\varepsilon n(4 + 2n/k)$$

for $j \in [2^k]$. Consequently, $\mathcal{E}$ will encode integers up to $m := (2^k - 1)\varepsilon n(4 + 2n/k) + 1 = (1 + o(1))2^k$. We then define data structure:

$$\mathcal{E}(\mu_j) := c^j \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n}$$

and

$$\mathcal{E}(\mu_j) := c^j \| 1^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 1^{\varepsilon n} \| \overline{\mathcal{B}(\rho_j, \beta^j)} \| 1^{\varepsilon n}$$

depending on the parity of $j$. Here, $\overline{\mathcal{B}(\rho_j, \beta^j)}$ is the bitwise complement of $\mathcal{B}(\rho_j, \beta^j)$. Note that each $\mathcal{E}(\mu_j)$ is $(1 + 4\varepsilon + 4\varepsilon n/k)n$ bits long. We infer that the code rate of $\mathcal{E}$ is $\mathcal{O}(\varepsilon)$-close to the capacity.

Next, we show that the Hamming distance between $\mathcal{E}(\mu_j)$ and $\mathcal{E}(\mu_{j+1})$ is $\varepsilon n(4 + 2n/k)$. Trivially, the consecutive zeros and ones contributes $3\varepsilon n$ bits of Hamming distance. Next, note that

$$c^{j+1} - c^j = g^{j+1}A - g^j A = (g^{j+1} - g^j)A = e^{\rho_j}A = A^{\rho_j}.$$

This is the $\rho_j$th row of $A$. By the construction (4), any row of $A$ contributes exactly $\varepsilon n$ bits of Hamming distance. Next, $\mathcal{B}(\rho_j, \beta^j)$ and $\mathcal{B}(\rho_{j+1}, \beta^{j+1})$ contributes an unknown amount of distance. But it is complement to the distance between $\mathcal{B}(\rho_j, \beta^j)$ and $\overline{\mathcal{B}(\rho_{j+1}, \beta^{j+1})}$. Therefore, the $\mathcal{B}$ part contributes exactly $2\varepsilon n^2/k$. In total, the Hamming distance is exactly $\varepsilon n(4 + 2n/k)$.

Now that the distance between consecutive milestones matches the Hamming distance, we can interpolate between them. In particular, for even $j$,

$$\begin{aligned}
\mathcal{E}(\mu_j) &\coloneqq c^j \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_j + \varepsilon n) &\coloneqq c^{j+1} \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_j + 2\varepsilon n) &\coloneqq c^{j+1} \| 1^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_j + 2\varepsilon n + d) &\coloneqq c^{j+1} \| 1^{\varepsilon n} \| \mathcal{B}(\rho_{j+1}, \beta^{j+1}) \| 0^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_j + 3\varepsilon n + d) &\coloneqq c^{j+1} \| 1^{\varepsilon n} \| \mathcal{B}(\rho_{j+1}, \beta^{j+1}) \| 1^{\varepsilon n} \| \mathcal{B}(\rho_j, \beta^j) \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_j + 3\varepsilon n + 2\varepsilon n^2/k) &\coloneqq c^{j+1} \| 1^{\varepsilon n} \| \mathcal{B}(\rho_{j+1}, \beta^{j+1}) \| 1^{\varepsilon n} \| \overline{\mathcal{B}(\rho_{j+1}, \beta^{j+1})} \| 0^{\varepsilon n}, \\
\mathcal{E}(\mu_j + 4\varepsilon n + 2\varepsilon n^2/k) &\coloneqq c^{j+1} \| 1^{\varepsilon n} \| \mathcal{B}(\rho_{j+1}, \beta^{j+1}) \| 1^{\varepsilon n} \| \overline{\mathcal{B}(\rho_{j+1}, \beta^{j+1})} \| 1^{\varepsilon n},
\end{aligned}$$

where $d$ is the Hamming distance between $\mathcal{B}(\rho_j, \beta^j)$ and $\mathcal{B}(\rho_{j+1}, \beta^{j+1})$.

## 2.3 Decoding $\mathcal{D}$

Suppose that we are given

$$c \| \phi \| B' \| \phi' \| B'' \| \phi''' \tag{5}$$

as the noisy version of $\mathcal{E}(x)$ for some $x \in [m]$, where

- $c \in \{0, 1\}^{1 \times (1+\varepsilon)n}$ is the noisy version of $c^j$, $c^{j+1}$, or anything in between,
- $\phi, \phi', \phi'' \in \{0, 1\}^{1 \times \varepsilon n}$ are the noisy version of the buffers, and
- $B', B'' \in \{0, 1\}^{1 \times 2\varepsilon n^2/k}$ are the noisy version of the $\mathcal{B}$ part.

We first apply Fathollahi and Wootters's decoder [4] to the second half of (5)

$$\phi \| B' \| \phi' \| B'' \| \phi'''.$$

Their decoder counts how many ones and zeros are in $\phi$, $\phi'$, and $\phi''$. This tells us which minor milestone we are at. We use this information to determine which of $B'$ or $B''$ is undergoing interpolation, and which is trustworthy. And then, we use the trustworthy one to recover the row index $\rho_j$ and the backup data $\beta^j$ (or $\rho_{j+1}$ and $\beta^{j+1}$ depending on if $x$ is past $\mu_j + 2.5\varepsilon n + d$ or not). From now on, we just call them $\rho$ and $\beta$.

We define the rollback function Roll: $\{0, 1\}^{(1+\varepsilon)n} \times [k] \times \{0, 1\}^{\varepsilon n} \to \{0, 1\}^{(1+\varepsilon)n}$ that overwrites messed-up bits using backup data. More precisely, $\mathrm{Roll}(c, \rho, \beta)$ will be the vector $c$ after replacing $c_{i_1}$ with $\beta_1$, $c_{i_2}$ with $\beta_2$, and so on, where $i_1, i_2, \ldots$ are the indices where $A^\rho$ has 1. Our claim is that, it does not matter if it is $c^j$, $0^{\varepsilon n}$, or $\mathcal{B}$ that is undergoing interpolation, $\mathrm{Roll}(c, \rho, \beta)$ will just look like the noisy version of $c^j$ or $c^{j+1}$, which can be decoded by the decoder of $\mathcal{C}$. This can be seen more clearly by considering three cases.

Case 1: $x$ is between $\mu_j$ and $\mu_j + \varepsilon n$. This is the stage where we are adding $A^\rho$ to $c^j$. In this case, the $\mathcal{B}$ part backs up the subvector of $c^j$ that is undergoing interpolation; $\mathrm{Roll}(c, \rho, \beta)$ would just be a noisy version of $c^j$ that can be decoded by $\mathcal{C}$.

Case 2: $x$ is between $\mu_j + \varepsilon n$ and $\mu_j + 2.5\varepsilon n + d$. This is the case where $B'$ is not trustworthy and so Fathollahi and Wootters's decoder will decode $B''$ to $(\rho_j, \beta^j)$. In this case, $\mathrm{Roll}(c, \rho_j, \beta^j)$ will be a noisy version of $c^j$ that can be decoded by $\mathcal{C}$.

Case 3: $x$ is between $\mu_j + 2.5\varepsilon n + d$ and $\mu_j + 4\varepsilon n + 2\varepsilon n^2/k$. This is the case where $B''$ is not trustworthy and so Fathollahi and Wootters's decoder will decode $B'$ to $\rho_{j+1}, \beta^{j+1}$. In this case, $\mathrm{Roll}(c, \rho_{j+1}, \beta^{j+1})$ will be a noisy version of $c^{j+1}$ that can be decoded by $\mathcal{C}$.

Examining these three cases, we can see that $\mathrm{Roll}(c, \rho, \beta)$ will always yield $c^j$ or $c^{j+1}$. With that, we can compute $g^j$ and $j$. Now that we know $x \in [\mu_j, \mu_{j+1}]$, it suffices to compare (5) with $\mathcal{E}(\mu_j), \ldots, \mathcal{E}(\mu_{j+1})$ and see which one minimizes the Hamming distance. The minimizer will be our best bet of $x$.

## 2.4 Complexity and tail estimation

The complexity of $\mathcal{E}$ and $\mathcal{D}$ is linear in $n$. This is because Gray's encoding, Gray's decoding, $\mathcal{B}$'s encoding, $\mathcal{B}$'s decoding, determining whether $\phi$, $\phi'$, and $\phi''$ are zeros or ones, determining whether $B'$ or $B''$ is trustworthy, and Roll are all linear in $n$.

The tail estimation (3) boils down to the following components.

- With probability $2^{-\Omega(n)}$, we obtain the wrong $j$, i.e., $x \notin [\mu_j, \mu_{j+1}]$.
- The guesswork of $x$ conditioned on correct $j$ has tail probability $2^{-\Omega(t)}$.

The first bullet point is a consequence of the error probability of $\mathcal{B}$ being $2^{-\Omega(\varepsilon n)}$, which is $2^{-\Omega(n)}$ as we fixed $\varepsilon$. The second bullet point relies on what minimizes the Hamming distance between (5) and $\mathcal{E}(\mu_j), \ldots, \mathcal{E}(\mu_{j+1})$. Such analysis has been done before [6, Lemma 3.7] [4, Lemma 13], and we do not repeat it here. This finishes the proof.

## References

1 Martin Aumüller, Christian Janos Lebeda, and Rasmus Pagh. Representing Sparse Vectors with Differential Privacy, Low Error, Optimal Space, and Fast Access. *Journal of Privacy and Confidentiality*, 12(2), November 2022. `doi:10.29012/jpc.809`.

2 Roni Con, Dorsa Fathollahi, Ryan Gabrys, Mary Wootters, and Eitan Yaakobi. Robust gray codes approaching the optimal rate, 2024. `arXiv:2406.17689`.

3 Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. *Journal of Privacy and Confidentiality*, 7(3):17–51, 2016. `doi:10.29012/jpc.v7i3.405`.

4 Dorsa Fathollahi and Mary Wootters. Improved construction of robust gray code, 2024. `arXiv:2401.15291`.

5 Venkatesan Guruswami and Piotr Indyk. Linear-Time Encodable/Decodable Codes With Near-Optimal Rate. *IEEE Transactions on Information Theory*, 51(10):3393–3400, October 2005. `doi:10.1109/TIT.2005.855587`.

6 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)*, Proceedings, pages 1050–1066. Society for Industrial and Applied Mathematics, January 2024. `doi:10.1137/1.9781611977912.40`.

7 OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences, 2024. Published electronically at `http://oeis.org`.