A Survey of Pseudorandom Codes

Miranda Christ
Massachusetts Institute of Technology*
mchrist@cs.columbia.edu

September 2026

Abstract

A pseudorandom code (PRC) is a cryptographic primitive combining pseudorandom encryption and error-correcting codes (Christ and Gunn, CRYPTO 2024). That is, any polynomial number of codewords are pseudorandom, and decryption tolerates a constant rate of bit-flips. Although this might seem like a natural combination of standard properties, it is a fundamentally new object: PRCs are black-box separated from many primitives, including one-way functions, public-key encryption, and even virtual black-box obfuscation (Döttling et al., Garg et al., TCC 2025).

Pseudorandom codes were introduced to build *watermarks for AI-generated content*. In particular, they imply watermarks that preserve the model's quality in a strong sense and withstand modification of the content. This relationship makes pseudorandom codes an inviting area for researchers: improvements to this simple theoretical object yield better watermarks. Beyond watermarks, PRCs have been used to construct error-correcting codes against computationally bounded channels (Lu et al., CRYPTO 2026), and techniques underlying them have led to new public-key encryption schemes (Döttling et al., CRYPTO 2026, Dyseryn et al. ASIACRYPT 2026).

This survey aims to equip the interested researcher with tools to make progress in the area. We detail the current landscape of definitions, techniques, and applications. A main contribution is introducing new tricks and assembling existing tricks to generically strengthen PRCs (e.g., from substitution-robustness to random-deletion-robustness). We also show natural weakenings of PRCs that can be constructed from OWFs or even unconditionally. We conclude with a number of concrete open questions.

*Much of this survey was completed while the author was a student at Columbia University.

Contents

1 Introduction **3**

2 Preliminaries **6**
2.1 Cryptography preliminaries 7

3 Warm-up: natural attempts **8**
3.1 A generic construction for a subconstant noise rate 9

4 Various definitions **10**
4.1 Pseudorandomness notions 11
4.2 Robustness notions 13
4.3 Soundness notions 16
4.4 Unified notions 16

5 Constructions **18**
5.1 Low-density parity check constructions 19
5.2 Local weak pseudorandom function construction 21
5.3 Planted hyperloop construction 22
5.4 On sparse planted structure 23
5.5 Subexponential security and permuted codes 24
5.6 Other constructions 26

6 Generic transformations **27**

7 Applications to watermarks **36**
7.1 LLM watermarks 38
7.2 Image watermarks 48
7.3 Video watermarks 50
7.4 Robust steganography 50

8 Open questions **50**
8.1 Solution to Question 7 54

1 Introduction

A *pseudorandom code (PRC)* is an encryption scheme with two additional properties: pseudorandomness and robustness. Pseudorandomness dictates that without a secret key, any polynomial number of ciphertexts are computationally indistinguishable from random.[1] Robustness requires that decryption succeeds even when a constant fraction of a ciphertext's bits are flipped. One can view a pseudorandom code as a natural combination of pseudorandom encryption and an error-correcting code.

Pseudorandom codes were introduced by Christ and Gunn [CG24] to watermark AI-generated content. A *watermarking scheme* alters the distribution of content (e.g., ChatGPT-generated text) to plant a hidden pattern that is discernible with a detection key. A canonical example of an LLM watermark involves randomly assigning each word the color red or green, then increasing the probabilities of green words. The detection key is this assignment of colors to words. The detector, knowing this assignment, takes a given piece of text and computes the fraction of green words; if significantly more than half of the words are green, the text is watermarked.

This scheme satisfies two required properties of watermarks. First, it is *sound*, meaning that any text fixed before the random red-green assignment has negligible probability of detection.[2] Second, it is *robust*, meaning that sufficiently high-entropy responses will be detected as watermarked, even if many (specifically, a constant fraction) of the words are changed; to remove the watermark, one must change enough green words to red. However, it lacks the third essential property, *quality*: the watermark influences the model to prefer words on the green list, introducing biases in the topics it discusses and causing it to make poorer word choices.

Although it is easy to see how specific schemes *lack* quality, formalizing quality might seem more difficult: if we could precisely define what it means for a paper to be good, the review process would be much simpler. Thankfully, we can avoid such philosophical questions by defining quality *relative to the original model*. Ideally, the watermark should preserve whatever quality the original model had. The beautiful notion of indistinguishability can be used to formalize this: a watermark is quality-preserving, termed *undetectable* [CGZ24], if the watermarked model is computationally indistinguishable from the original model by any algorithm making a polynomial number of queries. An undetectable watermark preserves any efficiently measurable notion of quality.

It is not hard to see that achieving all three of these properties, soundness, robustness, and undetectability, *requires* a pseudorandom code. To be more concrete, we define the core notion of a PRC below.[3]

[1] This is CPA-pseudorandomness, for the familiar reader.

[2] Provided that the detector computes the fraction of green words among the *distinct* words.

[3] This is what's called a *zero-bit PRC*, meaning that decryption can distinguish noisy codewords from unrelated strings, but this can easily be converted to a PRC that encrypts longer messages. A zero-bit PRC is sufficient for watermarks, which we will soon see.

Definition 1 (Pseudorandom code ([CG24]))**.** *A* pseudorandom code *is a tuple of p.p.t. algorithms* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$*.*

- $\mathsf{KeyGen}(1^\lambda) \to \mathsf{sk}$*: takes as input the security parameter and outputs a secret key.*
- $\mathsf{Encode}(\mathsf{sk}) \to c$*: is a highly randomized algorithm that outputs a codeword* $c \in \{0,1\}^n$*.*
- $\mathsf{Decode}(\mathsf{sk}, c') \to 1$ *or* $\perp$*: determines whether* c' *is close to a codeword, in which case it outputs 1; otherwise, it outputs* $\perp$*.*

A pseudorandom code must satisfy pseudorandomness, soundness, and robustness*:*

- *(Pseudorandomness). For all p.p.t. distinguishers D,*

$$\left| \Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [1 \leftarrow D^{\mathsf{Encode}(\mathsf{sk})}(1^\lambda)] - \Pr_{\mathcal{U}_n} [1 \leftarrow D^{\mathcal{U}_n}(1^\lambda)] \right| \leq \mathsf{negl}(\lambda),$$

  where the oracle $\mathcal{U}_n$ *outputs a fresh uniform string on each query.*

- *(Soundness). For any* $x \in \{0,1\}^n$*,*

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [1 \leftarrow \mathsf{Decode}(\mathsf{sk}, x)] \leq \mathsf{negl}(\lambda).$$

- *(Robustness). There exists some constant* $p \in (0, 1/2)$ *such that*

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [1 \leftarrow \mathsf{Decode}(\mathsf{sk}, BSC_p(c)) : c \leftarrow \mathsf{Encode}(\mathsf{sk})] \geq 1 - \mathsf{negl}(\lambda).$$

To see why pseudorandom codes are necessary for watermarks, consider repeatedly prompting a watermarked LLM to output a random binary string. Undetectability requires that these strings are pseudorandom. Robustness requires that detection succeeds even when a constant fraction of the outputs' bits are changed. Soundness requires that the detector outputs "false" on unrelated strings. This gives a pseudorandom code, where the secret key is the detection key, the codewords are the LLM outputs, and the decryption algorithm is the detection algorithm. While this establishes that pseudorandom codes are necessary, of course they are of limited interest unless they are actually useful.

Christ and Gunn showed that pseudorandom codes imply our desired LLM watermarks, and furthermore that they can be constructed under standard cryptographic assumptions (specifically, subexponential hardness of Learning Parities with Noise). Their watermarking scheme is quite simple in hindsight, using two key ingredients.

The first is that LLMs are "randomness-recoverable." That is, an LLM can be viewed as an algorithm $\mathsf{Generate}$ takes as input a prompt PROMPT and a random string $r \in \{0,1\}^n$, and uses these together to generate some content c (e.g., an image or text). Christ and Gunn proved that for autoregressive LLMs, there is an accompanying $\mathsf{Recover}$ algorithm that, given c, outputs an approximation of the randomness r used to generate it. That is, for $r \leftarrow \{0,1\}^n$ and $c \leftarrow \mathsf{Generate}(\text{PROMPT}; r)$, the Hamming distance between $\mathsf{Recover}(c)$ and r is at most εn for some constant $\varepsilon \in (0, 1/2)$. The value of ε depends on the entropy

of the output distribution given the prompt; if the prompt asks for a fixed response there is no dependence on r.

The second key ingredient is, of course, the pseudorandom code. Its application is simple: the watermark replaces the randomness used by the model with *pseudorandom codewords*. That is, let $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ be a PRC whose decoder tolerates an ε rate of errors. The watermark's secret key will be $\mathsf{sk} \leftarrow \mathsf{KeyGen}$. Every time $\mathsf{Generate}$ is queried with some prompt PROMPT, sample $r \leftarrow \mathsf{Encode}(\mathsf{sk})$ and output $\mathsf{Generate}(\text{PROMPT}; r)$. To detect whether the watermark is present in some content c, one uses randomness recovery and checks if the resulting string is close to a codeword:

$$\mathsf{Decode}(\mathsf{sk}, \mathsf{Recover}(c)) \stackrel{?}{=} 1.$$

The properties of the pseudorandom code correspond to desirable properties of the watermark:

- Robustness ensures that the watermark is *robust*. Suppose that c actually came from $\mathsf{Generate}(\text{PROMPT}; r)$ for a codeword $r \leftarrow \mathsf{Encode}(\mathsf{sk}, 1)$. Then $\mathsf{Recover}(c)$ is ε-close to r, and robustness of the PRC implies that Decode will output 1. Furthermore, it turns out that modifying a bit of c translates to modifying a bit of $\mathsf{Recover}(c)$. As a result, the watermark tolerates additional bit changes up to the decoder's error tolerance.

- Pseudorandomness ensures that the watermark preserves the *quality* of the content. In particular, since the watermark has just replaced the randomness of $\mathsf{Generate}$ with pseudorandomness, the watermarked model is *computationally indistinguishable* from the original model.

- A low false positive rate is ensured by a third property of PRCs, *soundness*, which requires that any fixed string decodes to 1 with only negligible probability (over choice of sk).

While this template was first introduced for watermarking LLMs, it works for any randomness-recoverable generative model, which turns out to be many of them. PRC-based watermarks have since been implemented for image models [GZS25] and video models [HLLL25, ZZJ+26] as well.

Applications aside, pseudorandom codes have emerged as interesting cryptographic objects in their own right. Döttling et al. and Garg et al. [DMR25, GGW25] showed that PRCs are black-box separated from many generic primitives, including one-way functions, public-key encryption, and even virtual black-box obfuscation. Pseudorandomness and robustness have been difficult to achieve together, and many works putting forth new constructions rely on unusual or entirely new assumptions [GM24, GG25, DJK+26, CGG+25, DFV26]. This challenge has led to interesting new ideas, both in constructing the codes themselves, and in choosing plausible yet useful assumptions. Recently, pseudorandom codes have been used for computationally-bounded error correction [LSW26], and we hope there are more cryptographic applications waiting to be discovered.

As this new area has grown, so has a body of folklore, definitions, and applications that are written down disparately or not at all. There are many variants of pseudorandom codes

one can consider: different error models, pseudorandomness against different distinguishers, stronger soundness notions. Some weaker variants can be constructed from only one-way functions; other seemingly stronger variants can be constructed using standard PRCs and some cute tricks. Many of these tricks, if recorded at all, appear only implicitly in a paper's proof. Finally, there is little written discussion of how altering the properties of the pseudorandom code affects the resulting watermark.

This survey. The primary goal of this survey is to facilitate research on pseudorandom codes. A key contribution is explicitly detailing definitional variants, transformations between them, and clarifying the landscape (Figure 3). We also identify parts of the landscape that are not yet well understood and formulate them into concrete open questions (Section 8).

Organization. After some preliminaries (Section 2), Section 3 builds intuition about why pseudorandom codes are difficult to construct. Section 4 details the various notions of pseudorandom codes. Section 5 gives a high-level overview of several constructions of pseudorandom codes, with the goal of conveying common techniques and challenges. It is not meant to be a comparison of the merits of the schemes. Section 6 presents transformations between the PRC variants from Section 4. Section 7 details how pseudorandom codes are applied to watermarks and presents related watermarks under a unified lens. Finally, in Section 8 we suggest interesting open questions.

2 Preliminaries

We use λ to denote the security parameter, and we typically use n to denote the codeword length. Throughout, we assume that λ and n are polynomially related. We say that a function $f(\cdot)$ is *negligible* in λ if $f(\lambda) = o(1/q(\lambda))$ for every polynomial q. We say a probability is *overwhelming* if it is at least $1 - f(\lambda)$ for some negligible function f.

Let $d_H : \{0,1\}^n \times \{0,1\}^n \to \mathbb{Z}_{\geq 0}$ denote Hamming distance, and let $d_E : \{0,1\}^* \times \{0,1\}^* \to \mathbb{Z}_{\geq 0}$ denote edit distance. We write $\mathrm{wt}(v)$ to denote the Hamming weight of a vector $v \in \{0,1\}^*$.

Let $\mathrm{BSC}_p : \{0,1\}^* \to \{0,1\}^*$ denote the binary symmetric channel which flips each bit of its input independently with probability $p \in (0,1)$. Let $\mathrm{BDC}_p : \{0,1\}^* \to \{0,1\}^*$ denote the binary deletion channel which deletes each bit of its input independently with probability $p \in (0,1)$.

For $\eta \in (0,1)$ we write $\mathrm{Ber}(\eta)$ to denote the Bernoulli distribution with parameter η. For $n \in \mathbb{N}$, we write $\mathrm{Ber}(n,\eta)$ to denote an n-bit vector of independent draws from $\mathrm{Ber}(\eta)$.

We use a convenient convention when applying permutations $\pi \in S_n$. For a string $x \in \{0,1\}^n$, we write $\pi(x)$ to denote

$$\pi(x) : x_{\pi(1)} || \ldots || x_{\pi(n)} \text{ and similarly } \pi^{-1}(x) := x_{\pi^{-1}(1)} || \ldots || x_{\pi^{-1}(n)}.$$

$\underline{\mathsf{SigForge}_{\mathcal{A},\Pi}(\lambda)}$
$(\mathsf{pk},\mathsf{sk}) \leftarrow \mathsf{Gen}(1^\lambda)$
$(\mathsf{m},\sigma) \leftarrow \mathcal{A}^{\mathsf{Sign}_{\mathsf{sk}}(\cdot)}(\mathsf{pk})$
Let $\mathcal{Q}$ denote the set of queries made by $\mathcal{A}$ to $\mathsf{Sign}_{\mathsf{sk}}(\cdot)$.
`return` $\mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{m},\sigma) \wedge \mathsf{m} \notin \mathcal{Q}$

Figure 1: The signature forgery experiment $\mathsf{SigForge}_{\mathcal{A},\Pi}(\lambda)$

2.1 Cryptography preliminaries

Pseudorandom function (PRF). Let $\mathcal{F} = \{F_{\mathsf{sk}} : \{0,1\}^{\ell_1(\lambda)} \to \{0,1\}^{\ell_2(\lambda)} \mid \mathsf{sk} \in \{0,1\}^\lambda\}$ be a family of functions. $\mathcal{F}$ is a *pseudorandom function (PRF)* if F_{sk} is efficiently computable and for all polynomial-time distinguishers D,

$$\left| \Pr_{\mathsf{sk}\leftarrow\{0,1\}^\lambda}\left[D^{F_{\mathsf{sk}}(\cdot)}(1^\lambda)=1\right] - \Pr_{f}\left[D^{f(\cdot)}(1^\lambda)=1\right] \right| \leq \mathsf{negl}(\lambda).$$

where f denotes a random function from $\{0,1\}^{\ell_1(\lambda)}$ to $\{0,1\}^{\ell_2(\lambda)}$.

Weak pseudorandom function (wPRF). Let $\mathcal{F} = \{F_{\mathsf{sk}} : \{0,1\}^{\ell_1(\lambda)} \to \{0,1\}^{\ell_2(\lambda)} \mid \mathsf{sk} \in \{0,1\}^\lambda\}$ be a family of functions. $\mathcal{F}$ is a *weak pseudorandom function (wPRF)* if F_{sk} is efficiently computable and for all polynomial-time distinguishers D,

$$\left| \Pr_{\mathsf{sk}\leftarrow\{0,1\}^\lambda}\left[D^{(\mathcal{U},F_{\mathsf{sk}}(\mathcal{U}))}(1^\lambda)=1\right] - \Pr_{f}\left[D^{(\mathcal{U},f(\mathcal{U}))}(1^\lambda)=1\right] \right| \leq \mathsf{negl}(\lambda).$$

where f denotes a random function from $\{0,1\}^{\ell_1(\lambda)}$ to $\{0,1\}^{\ell_2(\lambda)}$. The oracle notation means that on each query, D receives a random value $x \leftarrow \mathcal{U}$ and its evaluation, either $F_{\mathsf{sk}}(x)$ or $f(x)$.

Digital signature scheme. We use the definition of a digital signature from [KL07], with small modifications. A *digital signature scheme* is defined over a message space $\mathcal{M}$ and consists of polynomial-time algorithms $(\mathsf{Gen},\mathsf{Sign},\mathsf{Vrfy})$ such that:

- Gen : takes as input a security parameter 1^λ and outputs a public-private key pair $(\mathsf{pk},\mathsf{sk})$.
- Sign : takes as input a private key sk and a message $\mathsf{m} \in \mathcal{M}$. It outputs a signature σ, which we write as $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathsf{m})$.
- Vrfy : takes as input a public key pk, a message m, and a signature σ. It outputs 1 if the signature is valid and 0 otherwise. We write $b := \mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{m},\sigma)$.

It is required that except with negligible probability over $(\mathsf{pk},\mathsf{sk})$ output by $\mathsf{Gen}(1^\lambda)$, it holds that $\mathsf{Vrfy}_{\mathsf{pk}}(\mathsf{m},\mathsf{Sign}_{\mathsf{sk}}(\mathsf{m})) = 1$ for every $\mathsf{m} \in \mathcal{M}$.

Definition (Security of a digital signature scheme.)**.** A digital signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ is *existentially unforgeable under an adaptive chosen-message attack*, or just *secure*, if for all p.p.t. adversaries $\mathcal{A}$,

$$\Pr[\mathsf{SigForge}_{\mathcal{A},\Pi}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$$

where the experiment $\mathsf{SigForge}_{\mathcal{A},\Pi}(\lambda)$ is defined in Figure 1.

3 Warm-up: natural attempts

When attempting to construct a PRC, one might begin by observing that pseudorandomness and robustness are easily achieved individually. CPA-pseudorandom encryption is implied by one-way functions, and we have a wide range of error-correcting codes that are efficiently decodable from a constant error rate. Perhaps we can combine a CPA-pseudorandom encryption scheme ($\mathsf{Encrypt}$) and an error-correcting code ($\mathsf{ECC.Encode}$) to yield a pseudorandom code.

Given a message m, arguably the two most natural ways to construct a (even multi-bit) PRC encoder are:

Encryption first: Output $\mathsf{ECC.Encode}(\mathsf{Encrypt}(m))$. While the ciphertext $\mathsf{Encrypt}(m)$ is pseudorandom, applying the error-correcting code may introduce noticeable structure. At an extreme, if $\mathsf{ECC.Encode}$ is a repetition code, $\mathsf{ECC.Encode}(\mathsf{Encrypt}(m))$ will clearly not be pseudorandom regardless of what $\mathsf{Encrypt}(m)$ is. While the repetition code is a poor choice, one can verify that other choices of the code (e.g., an LDPC code) also introduce discernible structure.

Error-correction first: Output $\mathsf{Encrypt}(\mathsf{ECC.Encode}(m))$. Here, the final output is indeed pseudorandom due to $\mathsf{Encrypt}$. However, it may not be error-correcting: consider the scheme where $\mathsf{Encrypt}$ outputs $(r, \mathsf{PRF}(r) \oplus m)$. That is, the PRC encoder[4] outputs $(r, \mathsf{PRF}(r) \oplus \mathsf{ECC.Encode}(m))$ where $r \leftarrow\!\!\$\ \{0,1\}^\lambda$. To decrypt a (possibly noisy) ciphertext (r', z) one attempts to recover the one-time pad $\mathsf{PRF}(r')$ and computes $\mathsf{ECC.Decode}(z \oplus \mathsf{PRF}(r'))$. Observe that if $r' \neq r$, the one-time pad is entirely changed and decryption is not possible. Again, perhaps this encryption scheme was a poor choice, but by checking other examples one can observe that any choice of $\mathsf{Encrypt}$ seems to lose any robustness we got from $\mathsf{ECC.Encode}$.

We could come up with other ways to combine $\mathsf{ECC.Encode}$ and $\mathsf{Encrypt}$, but hopefully we have gained intuition that doing so is difficult: outputs of $\mathsf{ECC.Encode}$ are clearly structured, and outputs of $\mathsf{Encrypt}$ are brittle. Interestingly, two elegant works [GGW25, DMR25] showed that this general approach is likely doomed. These works show that essentially *all cryptographic primitives* one can think of (e.g., one-way functions, public-key encryption, and even virtual black-box obfuscation) are black-box separated from pseudorandom codes.

Therefore, we must resort to concrete assumptions that somehow allow us to infuse pseudorandomness and robustness simultaneously, rather than layering them on individually.

[4]This is by far the most common scheme that people suggest to me when they first learn about pseudorandom codes.

Before moving on to constructions from concrete assumptions, it is instructive to observe that generic constructions are possible if one relaxes the requirements, specifically robustness. Recall that standard PRCs require robustness to a constant rate of error. If one relaxes this *at all*, to any subconstant rate, one can construct PRCs from one-way functions.

3.1 A generic construction for a subconstant noise rate

Here we slightly adapt a generic construction sketched in the technical overview of [CG24] and fully analyzed by Ghentiyala and Guruswami [GG25]. It is robust to any subconstant noise rate and is pseudorandom assuming only the existence of one-way functions. For simplicity of exposition, we show only that there is an inverse polynomial gap between the probability of correct decoding and soundness error. This gap can then be amplified by repetition to achieve overwhelming probability of correct decoding and negligible soundness error; see [GG25] and Theorem 4.

Let $p = o(1)$ be the desired noise rate. Then the following is a zero-bit PRC with full pseudorandomness but weakened robustness and soundness:

Construction 1 (low-noise PRC [GG25])**.** Let $n := \min\{\lambda, (\log \lambda)/p\}$. Let $\mathcal{F} = \{F_{\mathsf{sk}} : \{0,1\}^n \to \{0,1\} \mid \mathsf{sk} \in \{0,1\}^\lambda\}$ be a pseudorandom function.

- $\mathsf{KeyGen}(1^\lambda)$: Output $\mathsf{sk} \leftarrow \{0,1\}^\lambda$.
- $\mathsf{Encode}(\mathsf{sk})$: Rejection sample $r \leftarrow \{0,1\}^n$, subject to $F_{\mathsf{sk}}(r) = 1$. If no such r is found after λ attempts, the encoder fails and outputs 1^n.
- $\mathsf{Decode}(\mathsf{sk}, z)$: Output 1 if $F_{\mathsf{sk}}(z) = 1$; otherwise, output $\perp$.

Claim (Robustness of Construction 1)**.** *Under the assumption that $\mathcal{F}$ is a PRF, there exists some polynomial $\mathsf{poly}(\lambda)$ such that*

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [\mathsf{Decode}(\mathsf{sk}, BSC_p(x)) = 1 : x \leftarrow \mathsf{Encode}(\mathsf{sk})] - \frac{1}{2} \geq 1/\mathsf{poly}(\lambda).$$

Proof. We will show that with probability at least $1/\mathsf{poly}(\lambda)$, the channel BSC_p imposes no errors, in which case decoding will succeed by construction. If there is nonzero error, the probability of decoding to 1 is negligibly close to 1/2 by PRF security.

Each of the $n \leq ((1/p) \log \lambda)$ bits in the codeword receives an error independently with probability p. Therefore, the probability that all bits are error-free is at least

$$(1-p)^{(1/p)\log \lambda} \geq e^{-\log \lambda}(1-p)^{\log \lambda}$$

where we use the fact that $(1 + x/n)^n \geq e^x(1 - x^2/n)$ for $n > 1, |x| \leq n$. Observe that this is at least inverse polynomial in λ.

Finally, we note that the encoder's rejection sampling fails with negligible probability by pseudorandomness of the PRF. □

Claim (Pseudorandomness of Construction 1)**.** *Construction 1 is pseudorandom under the assumption that $\mathcal{F}$ is a PRF.*

noise rate \ alphabet size	$\Theta(1)$	$\omega(1)$
$\Theta(1)$	✗ ([GGW25, DMR25])	✓ (Construction 1, adapted)
$o(1)$	✓ (Construction 1)	✓ (Construction 1)

Table 1: (Im)possibility of generic PRC constructions using only one-way functions. ✗ denotes a black-box separation.

Proof. Consider an alternate encoder that works as follows: it rejection samples tuples (r, b) where r is as before and $b \leftarrow \{0,1\}$ is a uniform bit, until it obtains a tuple where $b = 1$. It then outputs that r. Observe that this encoder's outputs are uniformly random strings. It remains to show that the actual encoder is computationally indistinguishable from this encoder.

This is implied by pseudorandomness of the PRF, provided that the PRF is never queried on the same r twice. To rule this out, consider any adversary making a polynomial number of queries q to Encode. The probability that any r is queried more than once is at most $(q\lambda)^2 \cdot 2^{-n}$, which is immediately negligible if $n = \lambda$. If $n = (\log \lambda)/p$, we have

$$(q\lambda)^2 \cdot 2^{-(1/p)\log \lambda} = (q\lambda)^2 \cdot \lambda^{-1/p}.$$

Since $p = o(1)$, the factor $1/p$ in the exponent is $\omega(1)$ and this probability is negligible.

Finally, we note that the encoder only fails and outputs all ones with negligible probability. □

Claim (Soundness of Construction 1)**.** *Let $s \in \{0,1\}^n$ be any fixed string. Under the assumption that $\mathcal{F}$ is a PRF,*

$$\left| \Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [\mathsf{Decode}(\mathsf{sk}, s) \neq \perp] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda).$$

Proof. This follows immediately from pseudorandomness of $\mathcal{F}$; otherwise, there would exist a (nonuniform) distinguisher for $\mathcal{F}$ with s hard-coded. □

Remark 1. *The above construction can be adapted to the setting where the alphabet Σ has size $\omega(1)$ and the noise rate is any constant c.* Encode *rejection samples a random string r from $\Sigma^{\log \lambda}$ subject to $F_{\mathsf{sk}}(r) = 1$. Since $\omega(1)^{\log \lambda}$ is superpolynomial, we only repeat a PRF query with negligible probability, and this is pseudorandom.*

We summarize the generic state of affairs in Table 1.

4 Various definitions

In this section, we taxonomize the many definitions surrounding pseudorandom codes.

Definition 2 (Syntax)**.** *A* pseudorandom code *is a tuple of p.p.t. algorithms* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$*:*

- KeyGen *is a randomized algorithm that takes as input the security parameter and outputs the key material (a secret key, and optionally a public key).*
- Encode *is necessarily randomized. It takes as input a key and a message, and outputs a codeword.*
- Decode *may be randomized or deterministic. It takes as input a secret key and a string, and attempts to determine whether that string encodes a message. It outputs either that message or* $\perp$.

This syntax varies slightly depending on several properties of the pseudorandom code.

1. (Secret- versus public-key.) In a **secret-key** PRC, KeyGen outputs only a secret key. This same secret key is used by Encode and Decode. In a **public-key** PRC, KeyGen outputs a public/secret-key pair. Encode uses the public key, and Decode uses the secret key.
2. (Alphabet.) Encode outputs a codeword in Σ^n, where Σ is the *alphabet* and n is the codeword length. Most commonly, we consider binary-alphabet PRCs, i.e., $\Sigma = \{0,1\}$. See Section 7.1 for discussion on why a binary alphabet is necessary for applications.
3. (Message length.) The PRC may be zero-bit, single-bit, or multi-bit. For a **zero-bit** PRC, the message space is simply $\{1\}$, so we often omit the message as an input to Encode, writing $\mathsf{Encode}(\mathsf{sk})$. The decoder then outputs either 1 or $\perp$, which allows us to distinguish between strings that are (possibly corrupted) codewords, and strings that are unrelated to the code.

   For a **single-bit** PRC, the message space is $\{0,1\}$, and we always specify which message is input to the encoder. The decoder now must distinguish between encodings of 0, 1, and unrelated strings (on which it outputs $\perp$).

   Multi-bit PRCs are defined similarly, where the message space is $\{0,1\}^\ell$ for $\ell > 1$.

   The *information rate* of a pseudorandom code is the ratio between the message length ℓ and the bit-length of the codeword.

For the remainder of this section, we write definitions for zero-bit secret-key binary PRCs unless otherwise specified.

4.1 Pseudorandomness notions

One could consider many different notions of pseudorandomness. All that appear in the literature thus far fall under the following general template:

Definition 3 (Pseudorandomness to a class of distinguishers)**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is* ε*-pseudorandom to a class of distinguishers* $\mathcal{D}$ *if for all* $D \in \mathcal{D}$*,*

$$\left| \Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [1 \leftarrow D^{\mathsf{Encode}(\mathsf{sk},\cdot)}(1^\lambda)] - \Pr_{\mathcal{U}} [1 \leftarrow D^{\mathcal{U}}(1^\lambda)] \right| \leq \varepsilon(\lambda).$$

Here, $D^{\mathcal{U}}$ means that the algorithm D receives access to an oracle that responds to each query with a freshly drawn uniform string in $\{0,1\}^n$.

Instantiations of this definition in the literature vary its parameters, whose values are presented in **purple bold**:

1. (Class of distinguishers.) Of course, we cannot hope to have pseudorandomness against unbounded algorithms allowed to query their oracles an unbounded number of times. Therefore, the class $\mathcal{D}$ is limited in either or both of the following ways:
   - Resources. $\mathcal{D}$ is often computationally bounded, limited to **p.p.t.** or **space-bounded** (e.g., $n^{1.5}$-space) algorithms. $\mathcal{D}$ may be computationally **unbounded**, in which case a restriction on the number of queries is required for pseudorandomness to be achievable.
   - Number of queries. Distinguishers in $\mathcal{D}$ may be restricted in terms of the number of queries they can make to their oracles. Typically, they can make an **arbitrary polynomial number** of queries. For weaker pseudorandomness, they may be allowed to make a **limited number of queries** $f(\lambda)$. This makes the problem of constructing PRCs significantly easier, as the length of the secret key can be a function of f.
2. (Distinguishing advantage.) The **distinguishing advantage** ε is most commonly a negligible function, but $o(1)$ has been considered as well.

For **standard pseudorandomness**, $\mathcal{D}$ is the set of all p.p.t. algorithms allowed to make an arbitrary polynomial number of queries to their oracles. The distinguishing advantage ε is required to be a negligible function. More precisely, recalling from Definition 1:

Definition 4 (Standard pseudorandomness)**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is pseudorandom if for all p.p.t. distinguishers* D*,*

$$\left| \Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)} [1 \leftarrow D^{\mathsf{Encode}(\mathsf{sk})}(1^\lambda)] - \Pr_{\mathcal{U}} [1 \leftarrow D^{\mathcal{U}}(1^\lambda)] \right| \leq \mathsf{negl}(\lambda).$$

For $f(\lambda)$-**bounded-sample pseudorandomness**, $\mathcal{D}$ is the set of *all* (including computationally unbounded) algorithms making at most $f(\lambda)$ queries to their oracles. The distinguishing advantage is negligible. More precisely, and departing a bit from the distinguisher language:

Definition 5 (Bounded-sample pseudorandomness)**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is* k*-bounded-sample-pseudorandom if the two distributions*

$$\{c_1, \ldots, c_k \leftarrow \mathsf{Encode}(\mathsf{sk}) : \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)\} \text{ and } \{r_1, \ldots, r_k \leftarrow (\{0,1\}^n)^k\}$$

are $\mathsf{negl}(\lambda)$*-close in statistical distance.*

On relaxed pseudorandomness. When used to construct watermarks, PRC pseudorandomness ensures that the watermark does not degrade the quality of the watermarked content in any downstream application: the watermarked model is computationally indistinguishable from the unwatermarked model. If we already have an application in mind—say we want to run an $O(n^{1.5})$-time algorithm A on some watermarked text—it might be tempting to use a PRC with pseudorandomness only against $O(n^{1.5})$-time distinguishers.

There is a significant issue with this, which we will see by being a bit more precise. To ensure that quality is preserved, we would want to argue that if we could distinguish between A(watermarked texts) and A(unwatermarked texts), we could break (weak) pseudorandomness of the PRC. Our distinguisher would take its input (either PRC codewords or random strings), run them through the model (to yield either watermarked or unwatermarked texts), then run A. Therefore, regardless of the final application A, we always need to fool distinguishers as powerful as the *model*. Unfortunately, running the model itself is extremely computationally intensive in practice.

Bounded-sample pseudorandomness seems to be a more useful relaxation, as many applications do use a limited amount of model-generated content. Furthermore, it is easy to obtain $\mathsf{poly}(\lambda)$-bounded-sample pseudorandomness for any polynomial, which we show in Theorem 11—this construction is even unconditional.

4.2 Robustness notions

Broadly speaking, robustness[5] considers a game in which an adversary (also called a *channel*) is given a codeword and optionally some further information about the code, and is allowed to modify this codeword. The adversary wins if it outputs a string that decodes to $\perp$. We say a pseudorandom code is robust to a class of channels $\mathcal{E}$ containing randomized algorithms $\mathcal{A} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ if no channel in $\mathcal{E}$ wins this game with non-negligible probability.

A bit more formally:

Definition 6 (General robustness for zero-bit secret-key PRCs)**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is* robust *to a class of channels* $\mathcal{E}$*, receiving auxiliary information described by* aux*, if for all* $\mathcal{A} \in \mathcal{E}$*,*

$$\Pr_{\substack{\mathsf{sk}\leftarrow\mathsf{KeyGen}(1^\lambda)\\ \alpha\leftarrow\mathsf{aux}(\mathsf{sk})}}[\mathsf{Decode}(\mathsf{sk}, c') = \perp : c \leftarrow \mathsf{Encode}(\mathsf{sk}, 1), c' \leftarrow \mathcal{A}(c, \alpha)] \leq \mathsf{negl}(\lambda).$$

This definition attempts to unify various definitions in the literature (e.g., robustness to p-bounded channels [CG24], adaptive robustness [AAC+25]), resulting in the slightly unusual introduction of aux. This function aux outputs some auxiliary information that the channel can use to craft its errors. As we will see below, aux may output sk itself (the "full-information" adversary), or an oracle implementing $\mathsf{Encode}(\mathsf{sk}, \cdot)$. So in a slight abuse of notation, α may be a string or an oracle. In another slight abuse of notation, we write

[5]For ease of explanation, we consider robustness for zero-bit PRCs here. For multi-bit PRCs, the adversary can specify a message m and receive a codeword from $\mathsf{Encode}(m)$. It wins if it outputs a string that decodes to any $m' \neq m$, including $\perp$.

channels with no auxiliary information (such as the binary symmetric channel) as randomized algorithms taking a single input, $\mathcal{A} : \{0,1\}^* \to \{0,1\}^*$.

Now, observe that if $\mathcal{E}$ is not sufficiently limited, robustness is impossible. For example, suppose that some channel $\mathcal{A}_{0^n} \in \mathcal{E}$ always outputs 0^n. By soundness, $\mathsf{Decode}(\mathsf{sk}, 0^n) = \perp$ with overwhelming probability, and this channel will nearly always win. Consequently, we always consider constrained classes $\mathcal{E}$. There are many reasonable choices of constraints, each yielding a different robustness notion.

The most natural constraint, which appears in all robustness variations and rules out $\mathcal{A}_{0^n}$, is that the channel's output must be close to its input. Within this constraint, there are many parameters we can vary: for example, the metric with respect to which closeness is defined, the closeness threshold, and the computational efficiency of the adversary.

Below, we taxonomize the parameters that are often varied to obtain different robustness notions. The parameter values are presented in **purple bold**.

1. (Type of modification.) The adversary is allowed to modify its given codeword in some way, for example, by substituting a bit for another. Types of modifications allowed include **substitutions**, **deletions**, **insertions**, and **edits** (which include substitutions, deletions, and insertions).

2. (Amount of modification.) Recall that to rule out the bad channel $\mathcal{A}_{0^n}$, we must bound the number of modifications the adversary makes. That is, on input $c \in \{0,1\}^n$ it is allowed to make pn modifications for some $\mathbf{p} \in (0,1)$. For random errors, p is typically the expected fraction of errors; for non-random errors, p is the maximum allowed fraction of errors. For substitutions, nontriviality requires $p \in (0, 1/2)$,[6] whereas for deletions $p \in (0,1)$ may be tolerable.

3. (Knowledge.) This determines how much the adversary knows about the codeword it is given and the entire code, and is described by aux in our definition. At one extreme, the adversary can only make **random** modifications.[7] This could mean several different things, including flipping each bit independently with some probability $q < p$, or applying a random fixed-weight-pn error vector. An **oblivious** adversary, perhaps the most standard notion, is a randomized algorithm applied to the codeword it is given as input. In both of these cases, the auxiliary information given to the channel is $\perp$; that is, $\mathsf{aux}(\mathsf{sk}) \to \perp$.

   One could consider an adversary that has additional information about the code beyond the single codeword it is given. An **adaptive** adversary can query the encoder (on messages of its choice for multi-bit PRCs) a polynomial number of times before choosing its output as a function of its input and these queries. An **adaptive with decoder** adversary can query a decoding oracle on chosen strings as well as the encoder. The auxiliary information enables this oracle access. For both of these notions of adaptivity, if the scheme is public-key one omits the encoder oracle access and simply gives the adversary the public key as input. A **full information** adversary is given the secret key as input, as well as the public key if applicable; that is, aux outputs $(\mathsf{sk}, \mathsf{pk})$.

[6] Consider the channel that changes 1s to 0s until it reaches pn changes; for an overwhelming fraction of strings it will output 0^n.

[7] Thus far, the random adversary has been considered only for deletions and substitutions.

4. (Runtime.) The adversary may be **computationally bounded** or **unbounded**.

For example, robustness to a rate p of random substitutions, from our core notion of a PRC (Definition 1, [CG24]) has the following parameters: (1) substitutions, (2) expected error rate p, (3) random, (4) does not matter as the errors are random. We recall the formal definition below:

Definition 7 (Robustness to random substitutions [CG24])**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is* robust *to a rate* $p \in (0, 1/2)$ *of random substitutions if*

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)}[\mathsf{Decode}(\mathsf{sk}, BSC_p(c)) = \perp : c \leftarrow \mathsf{Encode}(\mathsf{sk}, 1)] \leq \mathsf{negl}(\lambda).$$

Definition 8 (Robustness to random deletions [CG24])**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is* robust *to a rate* $p \in (0, 1)$ *of random deletions if*

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)}[\mathsf{Decode}(\mathsf{sk}, BDC_p(c)) = \perp : c \leftarrow \mathsf{Encode}(\mathsf{sk}, 1)] \leq \mathsf{negl}(\lambda).$$

For example, robustness to p-bounded channels as defined in [CG24] has the following parameters: (1) substitutions, (2) p, (3) oblivious, (4) unbounded.

Worst-case robustness refers to a full-information, unbounded adversary. We write the following worst-case definitions for a deterministic decoder; worst-case robustness has not yet been considered for randomized decoders.

Definition 9 (Robustness to worst-case substitutions)**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is* robust *to a rate* $p \in (0, 1/2)$ *of worst-case substitutions if*

$$\Pr_{\substack{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c \leftarrow \mathsf{Encode}(\mathsf{sk}, 1)}}[\exists c' : d_H(c, c') \leq pn \wedge \mathsf{Decode}(\mathsf{sk}, c') = \perp] \leq \mathsf{negl}(\lambda).$$

Robustness to worst-case substitutions has the following parameters: (1) substitutions, (2) p, (3) full information, and (4) unbounded.

Definition 10 (Robustness to worst-case edits [GM24])**.** *A pseudorandom code* $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ *is* robust *to a rate* $p \in (0, 1/2)$ *of worst-case edits if*

$$\Pr_{\substack{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda) \\ c \leftarrow \mathsf{Encode}(\mathsf{sk}, 1)}}[\exists c' : d_E(c, c') \leq pn \wedge \mathsf{Decode}(\mathsf{sk}, c') = \perp] \leq \mathsf{negl}(\lambda).$$

Adaptive robustness for single- or multi-bit PRCs in the secret-key setting is defined using the following security game.

$\mathcal{G}^{\mathsf{robust\text{-}sk}}_{\mathcal{A},\mathsf{PRC},p}(1^\lambda)$:

1. The challenger sets $\mathsf{transcript} = \emptyset$ and samples $\mathsf{sk} \leftarrow \mathsf{PRC.KeyGen}(1^\lambda)$.

2. The adversary is allowed to make encoding queries. For each encoding query m, the challenger responds with $c \leftarrow \mathsf{PRC.Encode}(\mathsf{sk}, m)$ and sets $\mathsf{transcript} = \mathsf{transcript} \cup \{(m, c)\}$.

3. The adversary sends the challenger $x \in \{0,1\}^n$.

4. The challenger computes $m' = \mathsf{PRC.Decode}(\mathsf{sk}, x)$ (which could be $\perp$).

5. If there exists $(m, c) \in \mathsf{transcript}$ such that $\mathrm{wt}(x \oplus c) \leq pn$ and $m \neq m'$, then the adversary wins; otherwise the adversary loses.

Definition 11 (Secret-key adaptive robustness [AAC+25])**.** *For $p \in (0, 1/2)$ we say that a secret-key pseudorandom code* PRC *is* adaptively p-robust *if, for any efficient adversary $\mathcal{A}$,*

$$\Pr[\mathcal{A} \textit{ wins } \mathcal{G}^{\mathsf{robust\text{-}sk}}_{\mathcal{A},\mathsf{PRC},p}(1^\lambda)] \leq \mathsf{negl}(\lambda).$$

Secret-key adaptive robustness has the following parameters: (1) substitutions, (2) p, (3) adaptive, and (4) computationally bounded.

Adaptive-with-decoder robustness is captured in unified security notions (presented later in Section 4.4), which combine pseudorandomness, soundness, and robustness.

4.3 Soundness notions

A common question motivated by watermarking applications is whether it is possible to strengthen soundness into some notion of unforgeability. Soundness (Definition 1) says that any fixed string x decodes to non-$\perp$ with negligible probability, *over choice of the secret key*: For any $x \in \{0,1\}^n$,

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)}[1 \leftarrow \mathsf{Decode}(\mathsf{sk}, x)] \leq \mathsf{negl}(\lambda).$$

However, this says nothing about an adversary that might observe several codewords and attempt to produce a string that is far from all observed codewords yet decodes to 1. "Unforgeability" of some sort would prevent producing this string, and perhaps imply that in watermark applications it is difficult for an adversary to watermark arbitrary content. Concretely, people wish to prevent attackers from writing horribly offensive text and embedding a reputable company's watermark.

It turns out that watermarks with this kind of unforgeability can be constructed from pseudorandom codes with standard soundness, by combining them with digital signatures [CG24, Section 7.4]. Probably for this reason, unforgeability of pseudorandom codes has not been explicitly considered much.

Unforgeability is, in a sense, implicitly considered in the unified notions of PRCs, *ideal* and *CCA security*, which we present next in Section 4.4.

4.4 Unified notions

Ideal security is a unified security notion of secret-key PRCs. It says that the PRC is indistinguishable from a superpolynomially large *random code*, to any efficient distinguisher with black-box access to encoding and decoding oracles. That is, this ideal random code responds to each encoding query m with a freshly drawn uniform string c.[8] For each decoding

[8] If the encoding oracle is queried at most a polynomial number of times, this behavior is equivalent to the encoder of a superpolynomially large random code.

query x, it checks whether any previous encoding response is within the decoding radius δ of x. If so, it responds with the m queried to create it; if not, it responds with $\perp$.[9]

$\underline{\mathcal{G}^{\mathsf{real}}_{\mathcal{A},\delta,\mathsf{PRC}}(1^\lambda)}$:

1. The challenger samples $\mathsf{sk} \leftarrow \mathsf{PRC.KeyGen}(1^\lambda)$.
2. The adversary is allowed to make encoding and decoding queries:
   - For each encoding query m, the challenger responds with $\mathsf{PRC.Encode}(\mathsf{sk}, m)$.
   - For each decoding query x, the challenger responds with $\mathsf{PRC.Decode}(\mathsf{sk}, x)$.
3. The adversary returns a bit $\hat{b}$.

$\underline{\mathcal{G}^{\mathsf{ideal}}_{\mathcal{A},\delta}(1^\lambda)}$:

1. The challenger sets $\mathsf{transcript} = \emptyset$.
2. The adversary is allowed to make encoding and decoding queries:
   - For each encoding query m, the challenger responds with a fresh random string $c \leftarrow \{0,1\}^n$ and sets $\mathsf{transcript} = \mathsf{transcript} \cup \{(m, c)\}$.
   - For each decoding query $x \in \{0,1\}^n$, the challenger responds with a random m such that $(m, c) \in \mathsf{transcript}$ and $\mathrm{wt}(c \oplus x) \leq \delta n$; if no such m exists, the challenger responds with $\perp$.
3. The adversary returns a bit $\hat{b}$.

Definition 12 (Ideal secret-key PRC security [AAC+25])**.** *We say that a secret-key pseudorandom code* PRC *satisfies* ideal δ-robust security *if, for any efficient adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathcal{G}^{\mathsf{real}}_{\mathcal{A},\delta,\mathsf{PRC}}(1^\lambda) = 1] - \Pr[\mathcal{G}^{\mathsf{ideal}}_{\mathcal{A},\delta}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda).$$

CCA-security is an analogous unified notion for public-key PRCs. It differs from ideal security in that now, the adversary receives the public encoding key. It can now use this public key to construct codewords itself, in addition to querying the encoding oracle as before. Because the adversary can now create codewords on its own, in the "ideal world" case, we can no longer respond to decoding queries only by examining the transcript of codewords previously returned by the encoding oracle. Therefore, we amend the decoder to first check the transcript, then answer according to the real decoder.

[9]For any polynomial number of encoding queries, all responses c will be at distance roughly $n/2$ from each other with overwhelming probability. Therefore, for any constant $\delta \in (0, 1/4)$ any x will be within decoding radius δ of at most one c.

$\mathcal{G}^{\mathsf{pub}}_{\mathcal{A},\delta,\mathsf{PRC}}(1^\lambda, b)$:

1. The challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PRC.KeyGen}(1^\lambda)$. It sends pk to $\mathcal{A}$.
2. The adversary is allowed to make encoding and decoding queries. Depending on the value of the bit b, the challenger responds.
   - If $b = 0$:
     - For each encoding query m, it responds with $\mathsf{PRC.Encode}(\mathsf{pk}, m)$.
     - For each decoding query x, it responds with $\mathsf{PRC.Decode}(\mathsf{sk}, x)$.
   - If $b = 1$:
     - For each encoding query m, the challenger responds with a fresh random string c and sets $\mathsf{transcript} = \mathsf{transcript} \cup \{(m, c)\}$.
     - For each decoding query x, the challenger responds with a random m such that $(m, c) \in \mathsf{transcript}$ and $\mathrm{wt}(c \oplus x) \leq \delta n$. If no such m exists, the challenger responds with $\mathsf{PRC.Decode}(\mathsf{sk}, x)$.
3. The adversary returns a bit $\hat{b}$.

Definition 13 (CCA security for PRCs)**.** *We say that a public-key pseudorandom code* PRC *satisfies* δ-CCA security *if, for any efficient adversary* $\mathcal{A}$*,*

$$\left|\Pr[\mathcal{G}^{\mathsf{pub}}_{\mathcal{A},\delta,\mathsf{PRC}}(1^\lambda, 0) = 1] - \Pr[\mathcal{G}^{\mathsf{pub}}_{\mathcal{A},\delta,\mathsf{PRC}}(1^\lambda, 1) = 1]\right| \leq \mathsf{negl}(\lambda).$$

5 Constructions

We now present several constructions of pseudorandom codes. We focus on those with *standard pseudorandomness* and *robustness to a constant rate of random substitutions*. One exception is the planted hyperloop PRC [GG25], which allows $o(1)$ distinguishing advantage but still considers p.p.t. adversaries. At the end of this section, we briefly discuss some constructions achieving weaker pseudorandomness and summarize all constructions in Table 2.

Throughout this section, for the sake of clarity, we simplify and weaken many constructions. In particular:

- In soundness, we analyze only random, not fixed, strings. That is, we analyze the probability
$$\Pr_{\substack{\mathsf{sk}\leftarrow\mathsf{KeyGen}(1^\lambda)\\ s\leftarrow\{0,1\}^n}}[\mathsf{Decode}(\mathsf{sk}, s) = 1].$$
One can easily convert a PRC with this notion of soundness into one with standard soundness by applying a random mask to all codewords. That is, the secret key includes $z \leftarrow \{0,1\}^n$ that is applied to all outputs of Encode, and is reversed before decoding.
- We focus on robustness to a constant rate $\varepsilon \in (0, 1/2)$ of random substitutions. This can be generically transformed into stronger robustness, which we discuss in more detail in Section 6.

- We allow constant soundness error and robustness error. That is, we require only that there is an inverse polynomial gap between

$$\alpha := \Pr_{\substack{\mathsf{sk}\leftarrow\mathsf{KeyGen}(1^\lambda)\\ s\leftarrow\{0,1\}^n}}[\mathsf{Decode}(\mathsf{sk},s)=1] \text{ and } \beta := \Pr_{\substack{\mathsf{sk}\leftarrow\mathsf{KeyGen}(1^\lambda)\\ c\leftarrow\mathsf{Encode}(\mathsf{sk})}}[\mathsf{Decode}(\mathsf{sk},\mathrm{BSC}_\varepsilon(c))=1]$$

  One can easily amplify constructions with such a gap to have negligible soundness error and overwhelming probability of correct decoding.

  One can draw $q = \Omega(\lambda/(\beta-\alpha)^2)$ independent secret keys, and have the encoder output the concatenation of q codewords, one sampled under each secret key. The decoder now checks the fraction of the q codewords that decode to 1 and accepts if it is at least $(\alpha+\beta)/2$.

  Ghentiyala and Guruswami [GG25] formalized this amplification theorem, which we restate in Theorem 4. They use a slightly different construction that draws q independent codewords under the same secret key.

5.1 Low-density parity check constructions

The first PRC construction was given by Christ and Gunn [CG24], using low-density parity check (LDPC) codes. Let $\eta \in (0, 1/2)$ be a constant noise rate.

LDPC-PRC: **Low-density parity check PRC** [CG24]

$\underline{\mathsf{KeyGen}(1^\lambda)}$:
Sample sparse $P \in \mathbb{F}_2^{r\times n}$ and pseudorandom $G \in \mathbb{F}_2^{n\times m}$ s.t. $PG = 0$ `//see variants`
Output $\mathsf{sk} = (P, G)$

$\underline{\mathsf{Encode}(\mathsf{sk})}$:
Sample $s \leftarrow \mathbb{F}_2^m$ and $e \leftarrow \mathrm{Ber}(n, \eta)$
Output $Gs \oplus e$

$\underline{\mathsf{Decode}(\mathsf{sk}, c')}$:
Output 1 iff $\mathrm{wt}(Pc') < r\left(1/2 - r^{-1/4}\right)$

Before discussing the scheme further, the assumptions that will be relevant to its pseudorandomness are:

Assumption 1 (LPN assumption)**.** *For $\eta \in (0, 1/2)$ and $g : \mathbb{N} \to \mathbb{N}$, the $\mathsf{LPN}_{g,\eta}$ assumption states that for every $n \in \mathbb{N}$ and every polynomial-time adversary $\mathcal{A}$,*

$$\left|\Pr_{\substack{G\leftarrow\mathbb{F}_2^{n\times g(n)}\\ x\leftarrow\mathbb{F}_2^{g(n)}\\ e\leftarrow\mathrm{Ber}(n,\eta)}}[\mathcal{A}(G, Gx\oplus e)=1] - \Pr_{\substack{G\leftarrow\mathbb{F}_2^{n\times g(n)}\\ u\leftarrow\mathbb{F}_2^{n}}}[\mathcal{A}(G,u)=1]\right| \le \mathsf{negl}(n).$$

For the next assumption, we will need the following distribution $\mathcal{D}_1(n, m, t)$:

1. Sample s to be a random t-sparse vector in $\{0,1\}^n$.
2. Sample a random matrix $G \in \mathbb{F}_2^{n \times m}$ subject to $s^T G = 0$.
3. Output G.

Let $\mathcal{D}_0(n, m(n))$ denote the uniform distribution over $\mathbb{F}_2^{n \times m(n)}$.

Assumption 2 (Planted XOR assumption)**.** *For $m, t : \mathbb{N} \to \mathbb{N}$, the $\mathsf{XOR}_{m,t}$ assumption states that for every $n \in \mathbb{N}$ and every polynomial-time adversary $\mathcal{A}$,*

$$\left| \Pr_{G \leftarrow \mathcal{D}_0(n,m(n))}[\mathcal{A}(G) = 1] - \Pr_{G \leftarrow \mathcal{D}_1(n,m(n),t(n))}[\mathcal{A}(G) = 1] \right| \leq \mathsf{negl}(n).$$

Christ and Gunn discuss two ways to sample the pair (P, G). In both variants, the parity-check matrix P has $\Theta(\log n)$ ones per row (hence the term "low-density") and $r = \Theta(n)$ rows.

1. **Random narrow G:** If G is sufficiently narrow, in particular $m = O(\log^2 n)$, it is close to truly random. That is, first sample P to be a random sparse matrix. Then sample a random G conditioned on $PG = 0$. This distribution of matrices G is $\mathsf{negl}(\lambda)$-close to uniform in statistical distance ([CG24, Lemma 9]).

   Because the width of G is only $O(\log^2 n)$, one must invoke *subexponential hardness* of LPN to show that samples from Encode are pseudorandom.

   This is probably considered the primary construction from [CG24].

2. **Pseudorandom wider G:** Under the planted XOR assumption [ASS+23], one can sample a *pseudorandom*, wider G, where $m = n^\varepsilon$ for $\varepsilon \in (0, 1)$. Now, outputs of Encode are pseudorandom under *standard* hardness of LPN; the tradeoff is that the planted XOR assumption is quite new.

   Ghentiyala and Guruswami later show that one can prove pseudorandomness under a slightly weaker variant of the planted XOR assumption, which says that a *noisy* version of G is pseudorandom [GG25, Assumption 5.3].

These two sampling variants affect only KeyGen, so we otherwise treat them as one scheme. We discuss the scheme's properties below.

- **Pseudorandomness:** LPN states that for a random G, any polynomial number of samples $(G, Gs \oplus e)$ are pseudorandom. Therefore, once we establish that G is (pseudo)random, we need only invoke LPN.

  In the first variant above, G is statistically close to random but very narrow, requiring subexponential hardness of LPN.

  In the second variant, the wider G is pseudorandom under the planted XOR assumption, allowing us to then use standard hardness of LPN.

- **Soundness:** except with negligible probability, P is full row rank. Then, for a random u, Pu is random and its weight is at least $r(1/2-r^{-1/4})$ with overwhelming probability.
- **Robustness:** suppose we have a noisy codeword $Gs \oplus e \oplus e'$. Observe that $P(Gs \oplus e \oplus e') = P(e \oplus e')$. Since e and e' are both sparse and random, $(e \oplus e')$ is also sparse with overwhelming probability, i.e. it has weight at most δn for a constant $\delta \in (0, 1/2)$.

  Let us analyze the inner product between a single row P_i and this vector $(e \oplus e')$. The expected inner product of a random $\Theta(\log n)$-sparse vector and a δn-sparse vector is $1/2 - 1/\mathsf{poly}(n)$. Choosing the sparsity of the parity check appropriately makes this bias well below the decoder's threshold of $(1/2 - r^{-1/4})$.

Note that the LDPC-PRC is actually public-key, as encoding requires only G, and LPN states that samples from Encode are pseudorandom even if G is known.

Alrabiah et al. [AAC+25] later showed that a slight variant of this scheme satisfies *adaptive* robustness: even an adversary with oracle access to Encode cannot find a low-weight error e' that causes a codeword to decode incorrectly. Their analysis first uses two facts: Encode needs only the generator matrix G, and there are superpolynomially many log-sparse vectors s such that $s^T G = 0$. This means that even if the adversary is given G, the actual parity check matrix P is hidden—any of the vectors s could constitute the rows of P. The analysis concludes by showing that any low-weight e' thwarts a random P formed from these s vectors with only negligible probability.

5.2 Local weak pseudorandom function construction

Golowich and Moitra [GM24] construct a PRC whose pseudorandomness rests on the existence of $O(\log n)$-local weak pseudorandom functions. That is, the pseudorandom function's output depends on only $O(\log n)$ bits of its input. Here, we present a variant of their construction that, for ease of presentation, is simplified in that it achieves only weak robustness and soundness. The full construction can be viewed as essentially applying the amplification transformation of Ghentiyala and Guruswami [GG25] (Theorem 4) to this simplified variant.

Below, wPRF is a weak PRF from $\{0,1\}^n \to \{0,1\}$.

$\mathsf{PRC}_{\mathsf{wPRF}}$: **PRC from a local weak PRF** [GM24]

$\underline{\mathsf{PRC}_{\mathsf{wPRF}}.\mathsf{KeyGen}(1^\lambda):}$
Output $\mathsf{sk} \leftarrow \{0,1\}^\lambda$

$\underline{\mathsf{PRC}_{\mathsf{wPRF}}.\mathsf{Encode}(\mathsf{sk}):}$
Sample $r \leftarrow \{0,1\}^n$
Output $c = (r, \mathsf{wPRF}_{\mathsf{sk}}(r))$

$\underline{\mathsf{PRC}_{\mathsf{wPRF}}.\mathsf{Decode}(\mathsf{sk}, c' = (r', b')):}$
Output 1 iff $b' = \mathsf{wPRF}_{\mathsf{sk}}(r')$

- **Pseudorandomness:** by pseudorandomness of the weak PRF, for any polynomial number of random r's, the bits $\mathsf{wPRF}_{\mathsf{sk}}(r)$ are computationally indistinguishable from random.[10]

- **Soundness:**
$$\Pr_{\substack{\mathsf{sk}\leftarrow\{0,1\}^\lambda \\ r\leftarrow\{0,1\}^n \\ b\leftarrow\{0,1\}}}[\mathsf{wPRF}_{\mathsf{sk}}(r) = b] = \frac{1}{2}.$$

- **Robustness:** suppose a codeword is subjected to a constant rate $\varepsilon \in (0, 1/2)$ of random substitutions. A nice fact from Boolean analysis states that for any τ-local function f,
$$\Pr_{\substack{r\leftarrow\{0,1\}^n \\ r'\leftarrow\mathrm{BSC}_\varepsilon(r)}}[f(r) = f(r')] \geq \frac{1}{2} + \frac{1}{2}(1-2\varepsilon)^\tau.$$
Since wPRF is $\tau = O(\log n)$-local, the decoder is $(\tau+1)$-local, and $(1-2\varepsilon)^{\tau+1}$ is at least inverse polynomial in n.

5.3 Planted hyperloop construction

Ghentiyala and Guruswami [GG25] observe that a recent public-key encryption scheme of Bogdanov et al. [BKR23] is robust to error, making it a PRC. The construction plants an $O(\log n)$-sparse relation in Goldreich's PRG [Gol11].

Goldreich's PRG $G_{H,P} : \{0,1\}^\lambda \to \{0,1\}^n$ is parameterized by a λ-node k-hypergraph H with n hyperedges, and a predicate $P : \{0,1\}^k \to \{0,1\}$. An input $s \in \{0,1\}^\lambda$ is interpreted as values at the nodes of the hypergraph, and the ith bit of the output is computed by applying P to the nodes in the ith hyperedge. Goldreich's PRG is often conjectured to be $o(1)$-indistinguishable from random when H is chosen uniformly at random and P is an appropriate predicate. Here, [GG25] uses $P(x_1, \ldots, x_5) = x_1 + x_2 + x_3 + x_4x_5$.

Bogdanov et al. [BKR23] specify a way to sample a random 5-hypergraph H with a planted $O(\log n)$-hyperloop $S \subseteq [n]$. In the context of Goldreich's PRG, this hyperloop enforces a relation between those indices of PRG outputs:

$$\Pr_{s\leftarrow\{0,1\}^\lambda}\left[\bigoplus_{i\in S} G_{H,P}(s)_i = 0\right] \geq \frac{1}{2} + \frac{1}{\mathsf{poly}(n)}.$$

Bogdanov et al. conjecture that their distribution of hypergraphs with planted hyperloops is $o(1)$-indistinguishable from a random H.

[10] Golowich and Moitra actually add a constant rate of noise to c in Encode, which allows them to show pseudorandomness based on a slightly weaker assumption: roughly speaking, that wPRF is pseudorandom when a constant rate of noise is added to its input and output. This decreases the rate of robustness by a constant.

Hyperloop: **Planted hyperloop PRC** [GG25]

Hyperloop.KeyGen(1^λ):
Sample a 5-hypergraph H with planted hyperloop S
Output $\mathsf{sk} = (H, S)$

Hyperloop.Encode(sk):
Sample $s \leftarrow \{0,1\}^\lambda$
Output $G_{H,P}(s)$

Hyperloop.Decode(sk, c'):
Output 1 iff $\bigoplus_{i \in S} c'_i = 0$

- **Pseudorandomness:** this follows from pseudorandomness of H and Goldreich's PRG. Because H is $o(1)$-indistinguishable from a random H, and a random H is known to admit efficient distinguishing attacks with inverse polynomial probability, this construction is only $o(1)$-pseudorandom.
- **Soundness:** for a random string x, $\Pr[\oplus_{i \in S} x_i = 0] = 1/2$.
- **Robustness:** by construction, we already have that noiseless codewords decode to 1 with probability at least $1/2 + 1/\mathsf{poly}(n)$. The noise sensitivity fact used by Golowich and Moitra, combined with locality of Decode, already suggests that this bias toward 1 should persist under random noise.[11]

A nice feature of this construction is that it is actually a public-key pseudorandom code, albeit with relaxed pseudorandomness. To generate a codeword, one simply runs Goldreich's PRG, which requires knowledge of the hypergraph H and fixed predicate P, but not the planted hyperloop. By assumption, outputs $G_{H,P}(s)$ are pseudorandom even if one knows H and P.

5.4 On sparse planted structure

We have just seen three fairly different constructions of pseudorandom codes: one using LDPC codes, one using a local weak PRF, and one using Goldreich's PRG. However, the same core idea underlies the robustness of all of these constructions: the decoder looks for some sparse structure planted in $\Theta(\log n)$ bits of the codeword. For LDPC-PRC, each row P_i of the parity check specifies a $\Theta(\log n)$-sized subset of codeword bits whose sum should be biased toward zero. For $\mathsf{PRC}_{\mathsf{wPRF}}$, a $\Theta(\log n)$-sized subset of codeword bits determines the weak PRF output. For Hyperloop, exactly like LDPC-PRC, a $\Theta(\log n)$-sized subset of codeword bits has a sum that is biased toward zero. In all of these constructions, robustness follows from the fact that with inverse polynomial probability, this structure is entirely untouched by a constant rate ε of noise: $(1-\varepsilon)^{\Theta(\log n)} = \mathsf{poly}(n)^{-1}$.

[11] Showing this formally requires a bit of analysis as codewords are PRG outputs rather than uniformly random; see [GG25].

While this sparse structure enables robustness, it also leads to quasipolynomial-time distinguishing attacks— in time $n^{\Theta(\log n)}$, an adversary can search over all possible $\Theta(\log n)$-sized subsets. While increasing the size of the sparse structure could prevent such attacks, unfortunately it loses robustness: $(1-\varepsilon)^{\omega(\log n)}$ becomes negligible. In applications so far, pseudorandomness is used only to guarantee that watermarks do not harm quality, so quasipolytime distinguishing attacks are not devastating. However, subexponential security would of course be far better, and furthermore, the fact that all constructions thus far were "barely" pseudorandom perhaps suggests that pseudorandom codes lie just on the edge of what is possible.

Christ et al. [CGG$^+$25] observe that in a sense, $\Theta(\log n)$-sparse structure is necessary for a constrained class of constructions. That is, suppose codewords take the form $(r, f(r))$ where $r \leftarrow\!\!\$\ \{0,1\}^n$, and $\mathsf{Decode}(r', b')$ computes $f(r') \stackrel{?}{=} b'$. Robustness implies that f is noise tolerant. It then follows from simple facts of Boolean analysis that f has an inverse-*quasipolynomial*-magnitude Fourier coefficient supported on at most $O(\log n)$ coordinates. That is, there is a parity on $\Theta(\log n)$ coordinates that has absolute correlation $1/n^{O(\log n)}$ with f.

Distinguishing attacks against $\mathsf{LDPC\text{-}PRC}$ in particular are thoroughly studied in [WWC$^+$25].

5.5 Subexponential security and permuted codes

If we want subexponential security, we somehow need the decoder to look at the entire codeword. A natural starting point is to use an error-correcting code whose symbols are t-wise independent:[12] if we take a random codeword, any t-sized subset of symbols are jointly uniformly random. For example, Reed-Solomon codes satisfy this property for t equaling the degree of the underlying polynomial, and are still efficiently decodable from a constant rate of error. If we set $t = n^{\Omega(1)}$, we could hope to prevent quasipoly-time distinguishing attacks; at the very least, the distinguisher can no longer brute-force search over small subsets.

Of course, such error-correcting codes still have noticeable structure; one can easily distinguish Reed-Solomon codewords from random strings. Christ et al. [CGG$^+$25] propose a construction that scrambles codewords to make them pseudorandom.[13] They conjecture that pseudorandomness holds as long as the underlying code has $n^{\Omega(1)}$-wise independence, and formalize this as the *permuted codes assumption*. When their construction is instantiated with Reed-Solomon codes, they observe that pseudorandomness is actually implied by an assumption previously introduced to construct doubly-efficient private information retrieval [BIPW17, CHR17].

We present their construction $\mathsf{PRC}_{\mathsf{pc}}$, restricted to binary codes for the ease of exposition, below. Here, C is a binary error-correcting code with blocklength n, dual distance $n^{\Omega(1)}$, and efficient decoding from a rate $\delta \in (0, 1/4)$, $\delta > \varepsilon$, of errors. We let $C.\mathsf{Decode}$ denote the efficient algorithm that, on input x, returns the unique $c \in C$ such that $d_H(x, c) \le \delta n$, or $\perp$ if no such c exists.

[12] In other words, for linear codes, the dual distance is greater than t.

[13] This template was actually presented in passing by Christ and Gunn [CG24, Section 4.2], but they described it as a "heuristic construction" and gave no security analysis.

$\mathsf{PRC}_{\mathsf{pc}}$: **Permuted codes [CGG$^+$25]**

$\underline{\mathsf{PRC}_{\mathsf{pc}}.\mathsf{KeyGen}(1^\lambda)}$:
$\sigma \leftarrow S_n$
Output $\mathsf{sk} = \sigma$

$\underline{\mathsf{PRC}_{\mathsf{pc}}.\mathsf{Encode}(\mathsf{sk})}$:
Sample $x \leftarrow C$
Let $x' \leftarrow \mathrm{BSC}_\varepsilon(x)$
Output $\sigma(x')$

$\underline{\mathsf{PRC}_{\mathsf{pc}}.\mathsf{Decode}(\mathsf{sk}, c')}$:
Let $y = \sigma^{-1}(c')$
Output 1 iff $C.\mathsf{Decode}(y) \neq \perp$

Note that this construction can be instantiated with many choices of the code C, including those with larger alphabets. However, for larger-alphabet codes, one must also permute each ith *symbol* using a different permutation $\pi_i \leftarrow S_\Sigma$. These permutations are then included in the secret key.[14] [CGG$^+$25] observe that these additional permutations are actually necessary; without them, there is a polynomial-time distinguishing attack when C is a Reed-Solomon code.

[CGG$^+$25] show that using folded Reed-Solomon codes, converted to binary after σ is applied, yields a binary PRC with robustness to a constant rate of *worst-case edits*. Giving a sufficiently tolerant decoder for these binarized codewords takes a bit of work, as a constant rate of errors to the codeword's binary representation translates to a far greater error rate over the larger alphabet, necessitating list recovery. For substitution-robustness, [CGG$^+$25] suggest Algebraic Geometry codes [HvL98] and Raw Reed-Solomon codes [SKS19].

- **Pseudorandomness:** this follows immediately from the permuted codes assumption [CGG$^+$25, Conjecture 3.1], which exactly says that outputs of Encode are pseudorandom.
- **Robustness:** robustness is inherited from $C.\mathsf{Decode}$.
- **Soundness:** distance of the code implies that a random point is far from all codewords.

Christ et al. also show that $O(\log n)$ samples from $\mathsf{PRC}_{\mathsf{pc}}.\mathsf{Encode}$ are $\mathsf{negl}(\lambda)$-close to random in statistical distance [CGG$^+$25, Corollary 4.3].

While $\mathsf{PRC}_{\mathsf{pc}}$ is secret-key, there is a clear route to making it public-key. The only secret information is the permutation σ. Since the underlying code is linear, a linear combination of σ-permuted codewords is another σ-permuted codeword. Therefore, one could hope to make the public key a bunch of samples from $\mathsf{Encode}(\mathsf{sk})$. In our new public-key scheme, one could encode by taking a random linear combination of these samples and adding noise. Unfortunately, the noise added by BSC_ε in Encode would be overwhelming.

[14] Notice that when $\Sigma = \{0, 1\}$, applying these alphabet-wise permutations is equivalent to applying a fixed random mask. This is useless for pseudorandomness since the distinguisher can add pairs of codewords, canceling out the mask at the cost of increasing the noise rate.

Another option is to make the public key a bunch of *noiseless* permuted samples $\sigma(x)$ for $x \leftarrow C$. This is exactly McEliece, which is known to be insecure for certain algebraic codes, including Reed-Solomon codes [CGG$^+$14], Algebraic Geometry codes [CCP17], and Raw Reed-Solomon Codes [DFV26]. Security of McEliece is most commonly conjectured for binary Goppa codes, whose noise tolerance is subconstant, making them unsuitable for pseudorandom codes.

Two remedies have been proposed. The first, by Döttling et al. [DJK$^+$26], hopes to prevent these attacks by adding a low rate ($O(1/\sqrt{n})$) of noise to the permuted codewords comprising the public key. They call this the *Strong McEliece with Noise* assumption [DJK$^+$26, Assumption 2]. Encode then takes an $O(\sqrt{n})$-sparse random linear combination of the public-key samples and adds a constant rate of noise. Pseudorandomness of ciphertexts then follows from pseudorandomness of the public key (implied by Strong Noisy McEliece) and $(1/\sqrt{n})$-noise LPN.

The second, by Dyseryn et al. [DFV26], involves multiplying the public key by a matrix whose inverse is sparse (specifically, has constant row Hamming weight). They conjecture that this public key is pseudorandom, and they call this the *Decisional Distorted McEliece Public Key* assumption [DFV26, Definition 14]. Encode multiplies a random vector by this public key and adds a constant rate of noise. Pseudorandomness of ciphertexts follows from pseudorandomness of the public key (implied by Distorted McEliece) and constant-noise LPN.

5.6 Other constructions

Some constructions can be viewed as applying generic transformations to the above constructions. Instead of presenting these constructions as standalone, we show the generic transformations in the next section.

In Table 2, we present a table with all constructions, including those obtained using generic transformations. One construction that is in the table but not discussed in detail is SSR, from [GG25, Section 6], which is pseudorandom only against space-bounded p.p.t. distinguishers. This pseudorandomness holds unconditionally. This construction also uses low-density (specifically $\Theta(\log n)$-sparse) parity checks.

These constructions are all incomparable, with the exception of those obtained by generic transformations (i.e., the deletion-robust $\mathsf{PRC}_{\mathsf{del}}$ is strictly stronger than the substitution-robust PRC used to build it). While $\mathsf{PRC}_{\mathsf{pc}}$ has the strongest robustness ($\Theta(1)$ worst-case edit robustness) and full pseudorandomness, it is only secret-key and relies on a new and nonstandard assumption. On the other hand, $\mathsf{LDPC\text{-}PRC}$ has only random substitution-robustness but is public-key and rests on arguably the most standard assumption of any construction, subexponential LPN. $\mathsf{Hyperloop}$ may be the weakest construction, as it achieves weak pseudorandomness and relies on the new planted hyperloop assumption; however, it is public-key and still widens the range of assumptions we can use to construct PRCs.

Table 2: Known PRC constructions and their guarantees.

Construction	Robustness	Pseudorandomness	Key	Assumption
LDPC-PRC [CG24]	$\Theta(1)$ random substitutions	Full	Public	Subexponential LPN
LDPC-PRC [CG24]	$\Theta(1)$ random substitutions	Full	Public	Standard LPN and weak planted XOR [GG25]
$\mathsf{PRC}_{\mathsf{wPRF}}$ [GM24]	$\Theta(1)$ random substitutions	Full	Secret	Log-local weak PRF
Hyperloop [GG25]	$\Theta(1)$ random substitutions	$o(1)$ distinguishing advantage	Public	Planted hyperloop [BKR23] and Goldreich's PRG
SSR [GG25]	$\Theta(1)$ random substitutions	$O(n^{3/2-\varepsilon})$-spc. PPT adversaries	Secret	Unconditional
PRC^{sharp} [AAC+25]	$\Theta(1)$ adaptive substitutions	Full	Secret	Pseudorandomness of LDPC-PRC
$\mathsf{PRC}_{\mathsf{pc}}$ [CGG+25]	$\Theta(1)$ worst-case edits	Full	Secret	Permuted codes/puzzles [BIPW17]
[DJK+26]	$\Theta(1)$ adaptive substitutions	Full	Public	Noisy McEliece [DJK+26] and $O(1/\sqrt{n})$-LPN
[DFV26]	$\Theta(1)$ adaptive substitutions	Full	Public	Distorted McEliece [DFV26] and standard LPN

Notes. "Full" pseudorandomness means that the encoding oracle is computationally indistinguishable from an oracle that returns a fresh uniformly random string on each query; this is the original definition from [CG24].

6 Generic transformations

Here, we describe several generic transformations from the literature. We state these theorems informally. First, in Theorems 1 and 2 we show that robustness to random substitutions can be converted into robustness to worst-case oblivious substitutions. Therefore, going forward we sometimes refer to "substitution robustness" without specifying whether we mean random or worst-case oblivious substitutions, as they are easily interchangeable.

Theorem 1 (Fixed-weight random substitutions to oblivious substitutions, implicit in [CG24])**.** *For any constant $p \in (0, 1/2)$, suppose that there exists* PRC *that is robust to every rate $q \leq p$ of random fixed-weight substitutions. We can transform this into* PRC' *that is robust to a rate p of oblivious substitutions and inherits the codeword length n and message length ℓ of* PRC*.*

Given $\mathsf{PRC} = (\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$, we construct $\mathsf{PRC}' = (\mathsf{KeyGen}', \mathsf{Encode}', \mathsf{Decode}')$ as follows:

- $\mathsf{KeyGen}'(1^\lambda)$: Sample $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, a random mask $r \leftarrow \{0,1\}^n$, and π a random permutation over $[n]$. Output sk, r, π.
- $\mathsf{Encode}'(\mathsf{sk}, r, \pi)$: Sample $c \leftarrow \mathsf{Encode}(\mathsf{sk})$ and output $\pi(c \oplus r)$.

- $\mathsf{Decode}'((\mathsf{sk}, r, \pi), c')$: Output $\mathsf{Decode}(\mathsf{sk}, \pi^{-1}(c') \oplus r)$.

Observe that to the oblivious channel, a single codeword is a uniformly random string due to the one-time pad. Therefore, it reveals no information about the permutation π. Its errors are then mapped to locations that are random from the channel's point of view, using π^{-1}.

Annoyingly, this transformation does not quite convert robustness to BSC_p into robustness to rate-p oblivious substitutions. Consider a PRC where the decoder outputs $\perp$ on every string that is *exactly* distance 1 from a codeword, and otherwise is worst-case robust. This PRC is robust to BSC_p, as the output of $\mathrm{BSC}_p(c)$ will only be distance 1 from c with negligible probability. However, the transformed PRC is not robust to the oblivious channel that makes one random substitution.

We can prevent this issue (and indeed all such pathologies) by having the decoder add further noise itself:

Theorem 2 (BSC_p to oblivious substitutions)**.** *For any constant $p \in (0, 1/2)$, suppose that there exists zero-bit PRC that is robust to BSC_p. We can transform this into zero-bit PRC'' that is robust to p-bounded oblivious substitutions and inherits the codeword length n of PRC.*

PRC'' is the same as PRC', but with a slightly modified decoder:

- $\mathsf{Decode}''((\mathsf{sk}, r, \pi), c')$: For $i \in \{0, \ldots, n\}$, let $e^{(i)} := 1^i || 0^{n-i}$, and let

$$y^{(i)} := \pi^{-1}(c' \oplus e) \oplus r.$$

  If for any i, $\mathsf{Decode}(\mathsf{sk}, y^{(i)}) = 1$, output 1. Otherwise, output $\perp$.

This decoder is the same as Decode' except that it changes a prefix one bit at a time, trying Decode as it goes. Some of these changes will reverse error added by the adversary, but eventually $c' \oplus e^{(i)}$ will be exactly distance pn from c, the codeword it came from. Now, by the same argument used for the previous transformation, the random permutation converts this into a random weight-pn error.

One can view BSC_p as first choosing a weight for its error vector and then choosing a random vector of that weight. BSC_p has probability mass $\Theta(1/\sqrt{n})$ over weight pn, and therefore Decode must be robust to random weight-pn error. This means that on $y^{(i)}$, Decode will output 1 with overwhelming probability.

This search over i increases soundness error by a factor of $(n+1)$, but this is still negligible.

Christ and Gunn show that any zero-bit PRC can be generically converted into a PRC that encodes longer messages, with an asymptotically favorable information rate:

Theorem 3 (Linear information rate [CG24, Construction 5])**.** *Suppose there exists a zero-bit PRC and an error-correcting code of information rate r, both robust to a rate p of random substitutions. For any $\varepsilon \in (0, p)$ and message length ℓ, we can transform this into an ℓ-bit PRC robust to a rate $(p - \varepsilon)$ of random substitutions with information rate approaching r as $\ell \to \infty$.*

The transformation is very simple. The idea is to encode a random nonce $r \in \{0,1\}^\lambda$ by concatenating λ codewords (encoding $r_i = 1$) or random strings (encoding $r_i = 0$). One then applies the regular error-correcting code ECC to the message and one-time pads this using $\mathsf{PRF}_{\mathsf{sk}}(r)$.

That is, to encode a message m, sample a random $r \in \{0,1\}^\lambda$. For all $i \in [\lambda]$ let $c_i \leftarrow \{0,1\}^n$ if $r_i = 0$ and $c_i \leftarrow \mathsf{PRC.Encode}()$ if $r_i = 1$. Output

$$c_1 || \dots || c_\lambda || \mathsf{PRG}(r) \oplus \mathsf{ECC.Encode}(m)$$

To decode, one recovers r by applying the PRC decoder to each c_i, reverses the one-time pad, and applies the decoder of ECC. As the message length increases, the information rate of this construction approaches that of ECC.

Theorem 4 (Amplification [GG25, Lemma 2.18])**.** *Suppose there exists a zero-bit PRC with weak substitution robustness and soundness: decoding recovers the message correctly with probability at least $\alpha(\lambda)$, and a random string decodes to 1 with probability at most $\delta(\lambda)$. If there is an inverse polynomial gap between α and δ, we can transform this weak PRC into a "full" PRC (i.e., with overwhelming probability of correct decoding and negligible soundness error).*

One concatenates one-time-padded codewords from the weak PRC.

Theorem 5 (Random substitutions to random deletions [CG24, Theorem 4])**.** *For any constants $p \in (0, 1/2)$ and $q \in (0, 1)$, there exists $\varepsilon > 0$ such that we can transform any PRC robust to a $(1/2 - \varepsilon)$ rate of substitutions into one robust to a rate p of random substitutions and q of random deletions. This transformation blows up the blocklength by a polynomial factor.*

Remark 2. *[CG24] proves only robustness to random deletions, as it was motivated by the "emoji attack" against LLM watermarks, in which an attacker asks the model to insert emojis throughout the text then deletes them. The transformation actually achieves robustness to random* insertions *as well as deletions (i.e., random edits).*

This transformation uses two key facts:

- One can sample a uniform (odd-length) binary string by first sampling $b \leftarrow \{0,1\}$ and then sampling a random string conditioned on its majority being b.
- For a random binary string x, randomly deleting a constant fraction of its bits preserves its majority with probability $1/2 + \Theta(1)$.

These facts enable the following encoder, for odd ℓ:

1. Sample a codeword $y \in \{0,1\}^n$ from the substitution-robust PRC.
2. For $i \in [n]$, sample $z_i \leftarrow \{x \in \{0,1\}^\ell : \mathrm{Maj}(x) = y_i\}$.
3. Output $z_1 || \dots || z_n$.

Pseudorandomness follows immediately from the first fact and pseudorandomness of the substitution-robust PRC. To decode, one partitions the given string into n approximately equal-length blocks, takes the majority of each block, and applies the substitution-robust PRC's decoder to this string. Observe that deletions will likely cause this partition to be misaligned from original blocks $z_1, \ldots, z_n$: for example, the first block may contain some bits that were originally in z_2. However, Christ and Gunn show that for sufficiently long ℓ, these guessed blocks will largely match the original z_i's, and furthermore their majority will likely be preserved under deletions. Blocks whose majority changes translate to errors imposed on the substitution-robust codeword y.

The above construction follows a more general template. Let $f : \{0,1\}^\ell \to \{0,1\}$ be any balanced function (i.e., $\Pr_{x \leftarrow \{0,1\}^\ell}[f(x) = 1] = 1/2$). Then we can construct an encoder:

1. Sample a codeword $y \in \{0,1\}^n$ from the substitution-robust PRC.
2. For $i \in [n]$, sample $z_i \leftarrow \{x \in \{0,1\}^\ell : \ f(x) = y_i\}$.
3. Output $z_1 || \ldots || z_n$.

The deletion-robust PRC simply takes f to be Maj.

The fact that f is balanced ensures pseudorandomness: if the y_i's were uniform and independent, the z_i's would be independent and drawn conditioned on $f(z_i)$ being a random bit. Since y is pseudorandom, so is z.

Robustness depends on the choice of f. Majority is a good choice for deletion robustness because it is fairly stable under deletions (from the deletion channel), and shifts and insertions (from the misaligned blocks). If we wanted robustness to some other error channel, we could instantiate this template with a different choice of f, provided that it is balanced.

Since this template allows us to improve the robustness of our PRC, a natural question is whether we can increase the rate of substitution robustness, from some rate $p \in (0, 1/2)$ to $p' > p$. Recall that our encoder starts by drawing a codeword y from the p-robust PRC. Then, for each $i \in [n]$, it samples z_i such that $f(z_i) = y_i$. Given a string $z' \leftarrow \mathrm{BSC}_{p'}(z)$, we decode by applying f to each block of z', hoping to recover the y_i's. Therefore, the most robust construction uses f maximizing

$$\Pr_{\substack{z_i \leftarrow \{0,1\}^\ell \\ z_i' \leftarrow \mathrm{BSC}_{p'}(z_i)}} [f(z_i') = f(z_i)].$$

The optimal choices of f are dictator functions (i.e., $f(x) = x_i$ for some fixed index $i \in [\ell]$), or their negations. Therefore, we might as well take $\ell = 1$, at which point this construction is exactly the same as the p-robust PRC we started with. This means that unfortunately, this template cannot boost the rate of substitution robustness. Finding a generic way to do so, or proving this is impossible, is an interesting open question.

Theorem 6 (Substitutions to edits [GM24, Theorem 4.1])**.** *There are constants $c_0, c_1 \geq 1$ such that the following holds. Suppose there exists a PRC with blocklength n robust to all $(1/2 - p)$-substitution-bounded channels, where $p < (10c_1)^{-1}$. Then we can transform this into a PRC with alphabet size $\lceil n \cdot c_0/p \rceil$, blocklength $m \leq n$, and robustness to any $(1 - c_1 p)$-edit-bounded channel.*

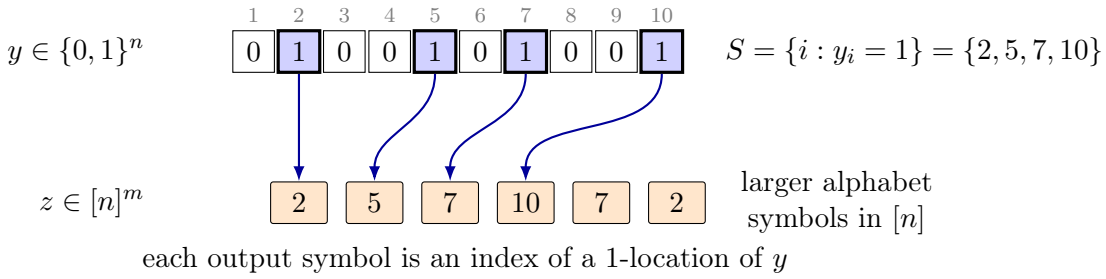

Figure 2: High-level idea behind the encoder of [GM24]

Golowich and Moitra [GM24] show how to transform any sufficiently substitution-robust binary PRC into a larger-alphabet PRC robust to a constant fraction of *edits*. Their key idea is to let each symbol of the edit-robust codeword represent an index of a '1' location of a corresponding substitution-robust codeword. As a result, the alphabet size of their edit-robust PRC blows up to be at least the blocklength of the substitution-robust PRC.

We present a simplification of their encoder, which is not pseudorandom but illustrates the general idea:

1. Sample a codeword $y \in \{0,1\}^n$ from the substitution-robust PRC.
2. For $i \in [m]$ (where $m < n$), sample $z_i \sim \text{Unif}(\{j \ : \ y_j = 1\})$.
3. Output $z = z_1 || \ldots || z_m$.

While any individual z_i is random, z overall is not pseudorandom as collisions among z_i's are more likely. The full construction of [GM24] carefully addresses this issue by deriving from y a tuple $t \in [n]^m$ whose set of distinct elements has small set difference with $\{i \in [n] : \ y_i = 1\}$. This small size is important, as elements of this set difference will effectively translate to errors. Finally, the edit-robust codeword is a random permutation of t.[15]

When considering robustness, observe that inserting a symbol i translates to setting the ith bit of the corresponding binary codeword to '1.' Similarly, deleting all symbols i translates to setting the ith bit to '0.' Therefore, substitution robustness of the binary PRC naturally yields edit robustness of the new PRC.

Note that a very recent paper [HLMZ26] shows how to convert a $(p > 1/4)$-substitution-robust PRC into an edit-robust PRC with a binary alphabet, tolerating any inverse polynomial rate of *edits*.

Actually, it is possible to achieve even slightly better edit robustness from only one-way functions: we can tolerate a $1/\omega(\log n)$ rate of worst-case edits. Note that this construction is secret-key, whereas the transformation of [HLMZ26] is public-key when applied to a public-key substitution-robust PRC.

Theorem 7 (Low-rate edit robustness from OWFs)**.** *Suppose one-way functions exist. Then for any $p = 1/\omega(\log n)$, there exists a PRC of blocklength n tolerating a rate p of worst-case edits.*

[15] Still, this is not the full scheme of [GM24], which improves robustness by further increasing the alphabet size, letting multiple symbols represent each index.

PRC codewords consist of $m = 2pn$ concatenated PRF input-output pairs. The inputs and outputs are of equal length $\ell = 1/(4p)$. That is, the construction uses a pseudorandom function $\mathsf{PRF} : \{0,1\}^\ell \to \{0,1\}^\ell$ and:

- $\mathsf{KeyGen}(1^\lambda)$: output $\mathsf{sk} \leftarrow \{0,1\}^\lambda$
- $\mathsf{Encode}(\mathsf{sk})$: sample $(x_1, \ldots, x_m) \leftarrow \{0,1\}^{m \cdot \ell}$. Output

$$c = (x_1, \mathsf{PRF}_{\mathsf{sk}}(x_1), \ldots, x_m, \mathsf{PRF}_{\mathsf{sk}}(x_m)).$$

- $\mathsf{Decode}(\mathsf{sk}, c')$: output 1 if $\mathsf{PRFsk}(c'[i : i + \ell - 1]) = c'[i + \ell : i + 2\ell - 1]$ for any $i \in [|c'| - 2\ell + 1]$. Otherwise, output $\perp$

Edit robustness follows simply from the fact that the channel can make at most pn edits. Since there are $m = 2pn$ input-outpu pairs, at least one pair must be edit-free, and the sliding-window decoder will observe that the PRF inputs and outputs match.

Pseudorandomness follows from PRF security, using the fact that the input length $\ell = 1/(4p) = \omega(\log n)$.[16] Therefore, with overwhelming probability the PRF will never be queried twice on the same input.

Theorem 8 (Adaptive robustness to idealness [AAC+25, Theorem 3])**.** *Suppose that there exists a single-bit secret-key PRC that is adaptively p-robust to substitutions. Then there exists a p-robust ideal pseudorandom code with a linear information rate.*

Adaptive robustness means that it is infeasible for an adversary with encoder query access to find $c \leftarrow \mathsf{Encode}(m)$ and c' that is p-close to c where $\mathsf{Decode}(c') \neq m$. Ideal security requires that PRC encoding and decoding oracles are computationally indistinguishable from "ideal" oracles implementing a random code with a superpolynomially sized codebook. That is, codewords are uniformly random strings, and the decoder checks if any codeword is p-close to the given string and returns its message if so; otherwise, it outputs $\perp$.

The adaptively robust PRC is nearly ideal: Pseudorandomness already implies that the PRC encoder looks like this ideal encoder, and adaptive robustness implies that the PRC decoder is at least as robust as the ideal decoder. The gap is that, somewhat counterintuitively, the PRC decoder may be *too* robust: strings more than p-far from a codeword may still decode to its message. In contrast, the ideal decoder should output $\perp$.

Alrabiah et al. [AAC+25] resolve this by "sharpening" the PRC so that its decoder fails everywhere beyond relative distance p. There are two main ideas here:

1. Ensure that any c' such that $\mathsf{Decode}(c') \neq \perp$ must be "associated with" exactly one $c \leftarrow \mathsf{Encode}(m)$ output by the encoding oracle.
2. Introduce a recovery algorithm that, given c' such that $\mathsf{Decode}(c') \neq \perp$, efficiently outputs its corresponding c and m.

[16] A common question when using short PRF inputs is whether a subexponentially secure PRF is required. Standard security of the PRF suffices, as the secret key is still λ bits. Of course, the resulting PRC will suffer from $2^{\ell/2}$-time attacks.

Once we have these two tools, we can check whether c and c' are p-close and output m if and only if this is the case.

Both ideas are solved simultaneously, using essentially the Fujisaki-Okamoto transform [FO99]. First, we transform the single-bit PRC into a multi-bit PRC $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ using essentially the transformation in Theorem 3. When queried on message m, the sharpened encoder outputs the following:

$$c \leftarrow \mathsf{Encode}(r||m||R_1; R_2),$$

where $r \leftarrow\!\!\$\ \{0,1\}^\lambda$ is drawn freshly for each encoding, and $R_1||R_2 = \mathsf{PRF}(r||m)$. The semicolon here indicates that the randomized encoding algorithm is using R_2 as its random coins.

The corresponding decoder, given some c', first applies Decode. If it obtains some $r'||m||R_1' \neq \perp$, it then verifies that both:

- The first component of $\mathsf{PRF}(r'||m)$ matches R_1', and
- c' is p-close to $\mathsf{Encode}(r'||m||R_1; R_2)$ where $R_1||R_2 \leftarrow \mathsf{PRF}(r'||m)$.

Only if these checks both pass does it output m.

Observe that this both ties c' to exactly one output of the encoder (the one that used that value of r), and allows us to recover this output.

Note that this transformation appears to be inherently secret-key, as it uses a pseudorandom function. It can be adapted to the public-key setting fairly straightforwardly, by replacing this pseudorandom function with a random oracle, achieving CCA security:

Theorem 9 (Adaptive robustness to CCA-security [AAC$^+$25, Theorem 4])**.** *Suppose that there exists a multi-bit public-key PRC that is adaptively p-robust to substitutions. Then in the random oracle model, there exists an $\Omega(p)$-robust CCA-secure pseudorandom code with linear information rate.*

Döttling et al. improve this result, giving a transformation in the standard model using a hinting PRG. Their construction is far more involved:

Theorem 10 (Adaptive robustness to CCA-security [DJK$^+$26, Theorem 9])**.** *Suppose that there exist a multi-bit public-key PRC that is robust to a constant rate of substitutions, and a secure hinting PRG. Then there exists a CCA-secure PRC.*

Next, we show how to transform any error-correcting code with polynomial dual distance into a "bounded-sample PRC:" a fixed polynomial number of codewords are jointly statistically close to random. We state this as a transformation for stylistic reasons, but this also shows unconditionally that poly-bounded-sample PRCs exist (since there exist efficiently decodable error-correcting codes with polynomial dual distance).

Theorem 11 (High-dual-distance ECC to poly-bounded-sample PRC)**.** *Let ECC be an efficiently decodable binary error-correcting code with codeword length n, dual distance $n^{\Omega(1)}$, and efficient decoding from a rate $p \in (0, 1/2)$ of random substitutions. There exists a constant $p' \in (0, 1/2)$ such that for any polynomial f, we can transform ECC into PRC that tolerates a rate p' of random substitutions and has $f(\lambda)$-bounded-sample pseudorandomness (Definition 5).*

Christ et al. [CGG$^+$25, Corollary 4.3] show that if ECC has polynomial dual distance, one can construct a $(\log n)$-bounded-sample PRC where the encoder samples a random codeword, randomly permutes it, and adds a constant rate of noise. The same random permutation is fixed as the secret key, and the noise is added independently to each sample. To decode, one reverses the permutation and applies $\mathsf{ECC.Decode}$. Since their encoder adds some noise, the robustness of their PRC is slightly reduced relative to ECC. We saw this construction in Section 5.5.

We now want to boost the number of samples that are random, from $\log n$ to $f(\lambda)$. Our idea is to use $k \gg f(\lambda)$ independent instances of the [CGG$^+$25] construction, which we denote $\mathsf{PRC}_{\mathsf{log}}$. Each time we encode, we will choose an instance at random and output one of its codewords. To decode, we will need to iterate over all k instances and check if the given string decodes to 1 under any of them. In more detail:

- $\mathsf{KeyGen}(1^\lambda)$: For $i \in [k]$, draw $\mathsf{sk}_i \leftarrow \mathsf{PRC}_{\mathsf{log}}.\mathsf{KeyGen}(1^\lambda)$.
- $\mathsf{Encode}(\{\mathsf{sk}_i\}_{i\in[k]})$: Sample $i \leftarrow [k]$. Output $c \leftarrow \mathsf{PRC}_{\mathsf{log}}.\mathsf{Encode}(\mathsf{sk}_i)$.
- $\mathsf{Decode}(\{\mathsf{sk}_i\}_{i\in[k]}, c')$: Output 1 iff $\mathsf{PRC}_{\mathsf{log}}.\mathsf{Decode}(\mathsf{sk}_i, c') = 1$ for some $i \in [k]$.

Observe that $f(\lambda)$-bounded-sample-pseudorandomness holds as long as we do not see more than $\log n$ samples from the same sk_i. The probability of seeing at least $\log n$ samples from instance i is at most:

$$\begin{aligned}\frac{\binom{f}{\log n} \cdot k^{f-\log n}}{k^f} &= \binom{f}{\log n} \cdot k^{-\log n} \\ &\le \left(\frac{ef}{\log n}\right)^{\log n} \cdot k^{-\log n} \\ &= O\left(n \cdot f^{\log n}\right) \cdot k^{-\log n}\end{aligned}$$

Taking $k = \lambda f$ suffices to make this expression negligible. A union bound over the k instances then implies that the probability of seeing $\log n$ samples from any sk_i is negligible.

Robustness follows immediately from robustness of $\mathsf{PRC}_{\mathsf{log}}$. Soundness follows similarly, though the soundness error increases by a factor of k due to a union bound over the k instances. This is still negligible as desired.

Note that this transformation incurs a factor of k blowup in the runtime of the decoder and the length of the secret key. This limits us to bounded-sample-pseudorandomness for a fixed polynomial number of samples. This is inherent: extending to arbitrarily many samples would yield an unconditionally secure PRC.

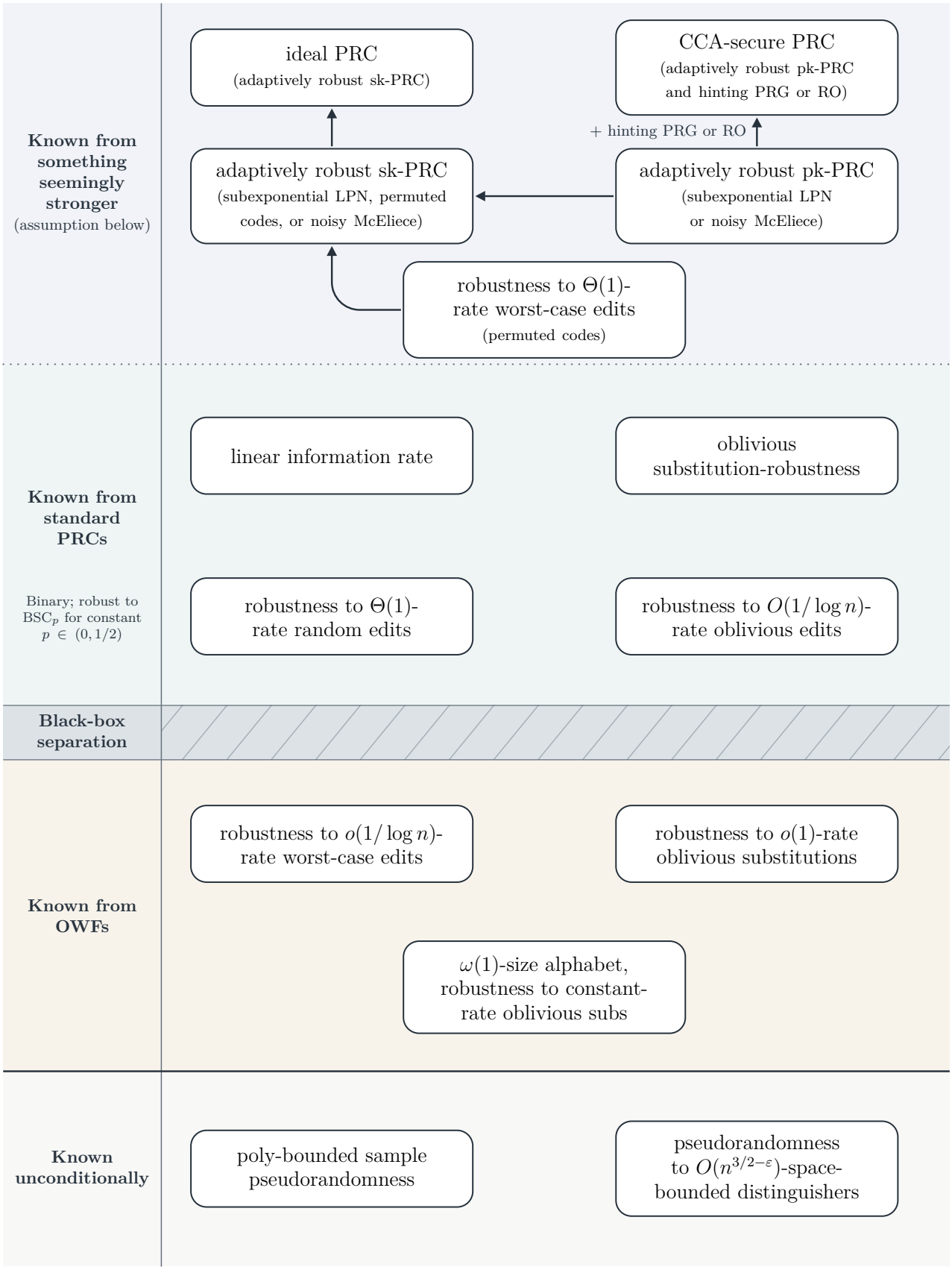

Figure 3: The landscape of pseudorandom codes.

7 Applications to watermarks

Christ and Gunn [CG24] present a general template for using PRCs to watermark randomized algorithms. The template applies to any algorithm taking, broadly, the following form.

Definition 14 (Randomness-recoverable algorithm)**.** *Let A be a randomized algorithm that, on input* PROMPT*, outputs some content c. A is* randomness-recoverable *if there exists a tuple of p.p.t. algorithms* $(\mathsf{Generate}, \mathsf{Recover})$ *such that:*

- $\mathsf{Generate}(\textit{PROMPT}; r)$ *takes as input a prompt* PROMPT *and randomness $r \in \{0,1\}^n$, and outputs some content c. It may supplement its random tape r with any amount of additional randomness.*
- $\mathsf{Generate}$ *implements the same randomized functionality as A; that is, for all prompts* PROMPT*,*
$$\mathsf{Generate}(\textit{PROMPT}; \mathcal{U}_n) \equiv A(\textit{PROMPT}; \mathcal{U}_n)$$
- *The accompanying randomness recovery algorithm* $\mathsf{Recover}$ *approximately recovers the randomness given the content. That is, for some constant $\varepsilon \in (0, 1/2)$, for all "nondegenerate" prompts* PROMPT*,*
$$\Pr_{r \leftarrow \{0,1\}^n}[d_H(\mathsf{Recover}(c), r) \le \varepsilon n : \ c \leftarrow \mathsf{Generate}(\textit{PROMPT}; r)] \ge 1 - \mathsf{negl}(\lambda).$$

Somewhat surprisingly, it turns out that many generative models (e.g., LLMs, image models) take this form, as we will see in more detail later in this section.

For now, let us see why PRCs are useful to watermark such algorithms. The idea is very simple: when running $\mathsf{Generate}$, replace the randomness $r \leftarrow \{0,1\}^n$ with *pseudorandomness* $r \leftarrow \mathsf{PRC.Encode}(\mathsf{sk})$, from a PRC tolerating a rate ε of oblivious errors. Given a piece of content $c \leftarrow \mathsf{Generate}(\text{PROMPT}; r)$, we can detect the watermark by checking

$$\mathsf{PRC.Decode}(\mathsf{sk}, \mathsf{Recover}(c)) \stackrel{?}{=} 1.$$

Notice that this gives us a secret-key watermark, where the same sk is used both to embed and detect the watermark. Here we present pseudocode for the watermark:

Watermark for $(\mathsf{Generate}, \mathsf{Recover})$ **using a pseudorandom code** PRC [CG24]

$\underline{\mathsf{Setup}(1^\lambda)}$:
Output $\mathsf{sk} \leftarrow \mathsf{PRC.KeyGen}(1^\lambda)$

$\underline{\mathsf{Wat}_{\mathsf{sk}}(\text{PROMPT})}$:
Sample $r \leftarrow \mathsf{PRC.Encode}(\mathsf{sk})$
Output $\mathsf{Generate}(\text{PROMPT}; r)$

$\underline{\mathsf{Detect}_{\mathsf{sk}}(c)}$:
Let $r' \leftarrow \mathsf{Recover}(c)$
Output true iff $\mathsf{PRC.Decode}(\mathsf{sk}, r') = 1$

The properties of the pseudorandom code immediately imply the desired properties of our watermark, presented informally here but defined fully for LLM watermarks in Section 7.1:

- **Undetectability:** The watermark merely replaces every random string with a PRC codeword. Therefore, by *pseudorandomness* of the PRC, the watermark is undetectable; for any polynomial number of adaptively chosen queries $x_1, \ldots, x_q$,

$$\begin{aligned}\{\mathsf{Generate}(x_1; r_1), \ldots, \mathsf{Generate}(x_q; r_q) : r_1, \ldots, r_q \leftarrow \{0,1\}^n\} \\ \approx_c \{\mathsf{Generate}(x_1; r_1), \ldots, \mathsf{Generate}(x_q; r_q) : r_1, \ldots, r_q \leftarrow \mathsf{Encode}(\mathsf{sk})\}\end{aligned}$$

- **Soundness:** *Soundness* of the PRC implies that for any fixed string c,

$$\Pr_{\mathsf{sk}}[\mathsf{Detect}(\mathsf{sk}, c) = \mathsf{true}] = \Pr_{\mathsf{sk}}[\mathsf{PRC.Decode}(\mathsf{sk}, \mathsf{Recover}(c)) = 1] \leq \mathsf{negl}(\lambda).$$

- **Completeness:** For any nondegenerate prompt PROMPT, *robustness* of the PRC implies that

$$\begin{aligned}&\Pr[\mathsf{Detect}(\mathsf{sk}, c) = \mathsf{true} : r \leftarrow \mathsf{PRC.Encode}(\mathsf{sk}), c \leftarrow \mathsf{Generate}(\text{PROMPT}; r)] \\ &\geq \Pr[d_H(\mathsf{Recover}(c), r) \leq \varepsilon n : r \leftarrow \mathsf{PRC.Encode}(\mathsf{sk}), c \leftarrow \mathsf{Generate}(\text{PROMPT}; r)] - \mathsf{negl}(\lambda) \\ &\geq 1 - \mathsf{negl}(\lambda).\end{aligned}$$

We emphasize that because Recover may be lossy, failing to recover r exactly, robustness of the PRC may be necessary for the watermark to be detectable *even in content output exactly by the watermarked algorithm*. One might ask that the watermark satisfies a stronger property of *robustness*, in that it persists even when the content is modified.

The robustness of the watermark varies depending on how modifying the content interacts with the error added by Recover. For example, if $c \leftarrow \mathsf{Generate}(\text{PROMPT}; r)$ is a piece of text from an LLM, one might fear that changing even one word of c, yielding c', would cause $\mathsf{Recover}(c')$ to be arbitrarily far from r. Fortunately, we will see later that careful choices of (Generate, Recover) cause this error to compose nicely and yield highly robust watermarks for LLMs and image models.

Necessity of pseudorandom codes. It is easy to see that if one can prompt the model to generate random binary strings, and one wants the watermark to remain even when a constant fraction of these strings' bits are changed, pseudorandom codes are required for undetectable watermarks. Undetectability implies that the watermarked model must output pseudorandom strings. Robustness implies that noisy versions of these strings must still be detected. Soundness implies that unrelated strings must not be detected.

However, if we are willing to compromise on undetectability, we can avoid the use of cryptography altogether. A weaker quality notion called *distortion-freeness*, defined by Kuditipudi et al. [KTHL24], requires only that a *single* watermarked sample is (statistically) indistinguishable from a single sample from the original model. Notice that this (Generate, Recover) framework *unconditionally* yields a distortion-free watermark: simply let the secret key be a fixed random r, and always use that r as input to Generate. To detect,

run $\mathsf{Recover}$ and check closeness to r. While this will harm diversity of the responses, a single response's quality will be preserved.

We can improve this diversity by using a k-bounded-sample PRC, which we showed exists unconditionally for any polynomial k in Theorem 11. This yields a watermark where any k responses are jointly distributed nearly identically to k responses from the original model (up to negligible statistical distance).

Discussion. One might wonder how we can *choose* $(\mathsf{Generate}, \mathsf{Recover})$ if we are given a particular model A. However, notice that for any $\mathsf{Generate}$, there are an incredible number of equivalent $\mathsf{Generate}'$: for example, $\mathsf{Generate}'(\text{PROMPT}; r)$ randomly permutes r to get r' and runs $\mathsf{Generate}(\text{PROMPT}; r')$. Since r' is uniform, $\mathsf{Generate}'(\text{PROMPT}; \mathcal{U}_n) \equiv \mathsf{Generate}(\text{PROMPT}; \mathcal{U}_n)$. In other words, there are many ways $\mathsf{Generate}$ can map each individual r to content c, while still generating the same distribution *when given a random* r. We will soon see concrete examples of different $\mathsf{Generate}$ describing the same LLM.

Another detail that may seem mysterious is the notion of "nondegenerate" prompts. Precisely what this means will vary by application (i.e., what kind of model we are watermarking). As an example, suppose that our algorithm is an LLM. Then a prompt "output the English alphabet in order" will yield an entirely deterministic response, and $\mathsf{Recover}$ cannot possibly succeed in recovering the randomness. Observe that success of $\mathsf{Recover}$ actually implies that the distribution $\mathsf{Generate}(\text{PROMPT}; r)$ has high entropy. Typically, nondegeneracy is some kind of high-entropy requirement.

Observe that a high-entropy requirement is not just required for the watermark to be detectable, but it is also desirable in practice. If a watermark is used to detect plagiarism, responses to many deterministic prompts such as our alphabet example *should not* be watermarked.

We now describe the applications of this watermarking framework to specific models. Along the way, we will see that other watermarks ([KTHL24, CGZ24]) can be viewed as slight modifications of this framework.

7.1 LLM watermarks

A *large language model* operates over a *token alphabet* $\mathcal{T}$, where each token represents a unit of language such as a partial word, character, or punctation mark (e.g., "the," "ing"), or the end of the response ("`done`"). The randomized generation algorithm takes as input a prompt $\text{PROMPT} \in \mathcal{T}^*$ and outputs a response $\mathsf{t} \in \mathcal{T}^*$.

The LLM watermarking literature focuses primarily on *autoregressive* LLMs, which generate responses in a highly structured way. An autoregressive LLM generates its response iteratively, shown below, using a neural network $\mathsf{Model} : \mathcal{T}^* \to \Delta(\mathcal{T})$. Model is a deterministic function that takes as input a sequence of tokens and outputs a probability distribution over $\mathcal{T}$. We use $\overline{\mathsf{Model}}$ to denote the randomized algorithm that takes as input a prompt and generates a response, using Model.

$\overline{\mathsf{Model}}(\textsc{prompt})$: generate a response to a prompt $\textsc{prompt}$
Initialize $i = 0$, $\mathsf{t}_0 = \perp$
While $\mathsf{t}_i \neq \texttt{done}$:
    $i = i + 1$
    Let $\mathsf{p}_i \leftarrow \mathsf{Model}(\textsc{prompt}, \mathsf{t}_1, \ldots, \mathsf{t}_{i-1})$
    Sample $\mathsf{t}_i \sim \mathsf{p}_i$ `//sampling step` $(*)$
Output $\mathsf{t} = (\mathsf{t}_1, \ldots, \mathsf{t}_i)$

An LLM watermark modifies the generation process to embed a detectable signal in the text. Nearly all LLM watermarks, and all we will see in this chapter, modify only the sampling step $(*)$ in the pseudocode above. As nice consequences, the neural network can be treated as a black box in the analysis, and a watermark can be added to an existing model without any further training.

7.1.1 Watermark definitions

We recall the formal definition of an LLM watermarking scheme from [CGZ24]:

Definition 15 (LLM watermarking scheme [CGZ24])**.** *A* watermarking scheme *for an autoregressive model* Model *over* $\mathcal{T}$ *is a tuple of polynomial-time algorithms* $\mathcal{W} = (\mathsf{Setup}, \mathsf{Wat}, \mathsf{Detect})$ *where:*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{sk}$ *outputs a secret key, with respect to a security parameter* λ*.*
- $\mathsf{Wat}_{\mathsf{sk}}(\textsc{prompt})$ *is a randomized algorithm that takes as input a prompt* $\textsc{prompt}$ *and generates a response in* $\mathcal{T}^\star$*.*
- $\mathsf{Detect}_{\mathsf{sk}}(\mathsf{t}) \to \{\mathsf{true}, \mathsf{false}\}$ *is an algorithm that takes as input a sequence* $\mathsf{t} \in \mathcal{T}^\star$ *and outputs* true *or* false*.*

A watermarking scheme should have the following properties:

- *Soundness*: independently generated text should not be detected as watermarked.
- *Completeness*: sufficiently high-entropy responses from the watermarked model should be detected.
- *Undetectability*: the watermark should not degrade the quality of the model; in particular, $\overline{\mathsf{Model}}$ and $\mathsf{Wat}_{\mathsf{sk}}$ should be computationally indistinguishable.

We present the formal definitions, from [CGZ24], below.

Definition 16 (Soundness)**.** *A watermarking scheme* $\mathcal{W}$ *is* sound *if for every security parameter* λ *and token sequence* $\mathsf{t} \in \mathcal{T}^\star$ *of length* $\mathsf{poly}(\lambda)$*,*

$$\Pr_{\mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda)}[\mathsf{Detect}_{\mathsf{sk}}(\mathsf{t}) = \mathsf{true}] \leq \mathsf{negl}(\lambda).$$

Definition 17 (Completeness)**.** *A watermarking scheme $\mathcal{W}$ is $b(L)$-*complete *if for every security parameter λ and prompt PROMPT of length $\mathsf{poly}(\lambda)$,*

$$\Pr_{\substack{\mathsf{sk}\leftarrow\mathsf{Setup}(1^\lambda)\\ \mathsf{t}\leftarrow\mathsf{Wat}_{\mathsf{sk}}(\mathit{PROMPT})}}[\mathsf{Detect}_{\mathsf{sk}}(\mathsf{t})=\mathsf{false}\ \textit{and}\ H_e(\mathsf{Model},\mathit{PROMPT},\mathsf{t})\geq b(\operatorname{len}\mathsf{t})]\leq\mathsf{negl}(\lambda).$$

Here, H_e is a notion of *empirical entropy*:

$$H_e(\mathsf{Model},\mathrm{PROMPT},t)=-\log\Pr[\overline{\mathsf{Model}}(\mathrm{PROMPT})=t].$$

The function $b(\cdot)$ specifies the entropy threshold making the watermark detectable.

Definition 18 (Undetectability)**.** *A watermarking scheme $\mathcal{W}=(\mathsf{Setup},\mathsf{Wat},\mathsf{Detect})$ is* undetectable *if for every security parameter λ and all polynomial-time distinguishers D,*

$$\left|\Pr[D^{\mathsf{Model},\overline{\mathsf{Model}}}(1^\lambda)\to 1]-\Pr_{\mathsf{sk}\leftarrow\mathsf{Setup}(1^\lambda)}[D^{\mathsf{Model},\mathsf{Wat}_{\mathsf{sk}}}(1^\lambda)\to 1]\right|\leq\mathsf{negl}(\lambda),$$

where the notation $D^{\mathcal{O}_1,\mathcal{O}_2}$ means that D is allowed to adaptively query both $\mathcal{O}_1$ and $\mathcal{O}_2$ with arbitrary prompts.

Another desirable property is *robustness*: not only should the watermark be detectable in a high-entropy text, but it should also withstand modification of that text. Deciding how much and what kind of modification to withstand is subtle. For example, one cannot hope for the watermark to survive arbitrary modification by polynomial-time algorithms: consider the algorithm that always outputs the English alphabet, in order. As we saw earlier, soundness *requires* that this text is not watermarked. And taking a step back, for many uses of the watermark (e.g., to detect plagiarism), sufficiently significant modifications of the text *should not* be watermarked.

So far, the literature has focused on notions of robustness that are relatively easy to define, analyze, and understand. They include robustness to a bounded number of token *substitutions*, *insertions*, and *deletions*. Notice that these notions correspond directly to the robustness notions of pseudorandom codes. Below, we define robustness of a watermark mirroring our general definition of robustness for pseudorandom codes.

Definition 19 (Robustness)**.** *A watermarking scheme $\mathcal{W}=(\mathsf{Setup},\mathsf{Wat},\mathsf{Detect})$ is $b(L)$-*robust *to a class of channels $\mathcal{E}$ if for all $\mathcal{A}\in\mathcal{E}$, for every prompt PROMPT,*

$$\Pr_{\substack{\mathsf{sk}\leftarrow\mathsf{Setup}(1^\lambda)\\ \mathsf{t}\leftarrow\mathsf{Wat}_{\mathsf{sk}}(\mathit{PROMPT})}}\left[\mathsf{Detect}_{\mathsf{sk}}(\mathcal{A}(\mathsf{t}))=\mathsf{false}\ \textit{and}\ H_e(\mathsf{Model},\mathit{PROMPT},\mathsf{t})\geq b(\operatorname{len}\mathsf{t})\right]\leq\mathsf{negl}(\lambda).$$

For example, $\mathcal{E}$ might be the class of all channels that change up to 1/3 of their inputs' tokens; robustness to $\mathcal{E}$ would be called *substitution-robustness*.

It is worth mentioning a more sophisticated kind of *semantic* robustness. All of the above notions are syntactic, in that they consider token-level modifications. What about an adversary that changes every single token, but preserves the meaning of the text? No kind of syntactic robustness guarantees that the watermark remains after such a modification, but

in principle such robustness seems possible. If an LLM were used to write a story, perhaps one could embed the watermark in high-level plot choices, character traits, and themes.

We do not yet have an undetectable and semantically robust LLM watermark. However, we *do* have an undetectable and semantically robust watermark for image models, by Gunn et al. [GZS25], discussed further in Section 7.2. There, the watermark is embedded in the latent feature vector of an image. The watermark therefore remains even if every pixel is changed, as long as the latent features are largely preserved.

Therefore, semantically robust undetectable watermarks for LLMs seem entirely possible. We highlight this as an open question (Section 8) in Section 8.

7.1.2 Detailed constructions

Recall that in order to watermark a randomized algorithm $\overline{\mathsf{Model}}$ using the framework of Christ and Gunn [CG24], all we need to do is specify a corresponding tuple $(\mathsf{Generate}, \mathsf{Recover})$.

For simplicity's sake, Christ and Gunn do so in two steps. They first use a transformation shown in [CGZ24] that converts any LLM $\overline{\mathsf{Model}}$ over $\mathcal{T}$ into an equivalent $\overline{\mathsf{Model}}'$ that operates over a *binary* token alphabet $\mathcal{T}' = \{0, 1\}$. The details of the binary transformation can be found in [CGZ24, Section 4.1] and [CG24, Page 48]. After reducing to binary, Christ and Gunn show the following $(\mathsf{Generate}, \mathsf{Recover})$ for any LLM $\overline{\mathsf{Model}}'$ over $\{0, 1\}$:

$(\mathsf{Generate}, \mathsf{Recover})$ for binary language models [CG24]

$\underline{\mathsf{Generate}(\text{PROMPT}; r)\text{:}}$
While $\mathsf{t}_i \neq \texttt{done}$:[a]
  $i = i + 1$
  Let $\mathsf{p}_i \leftarrow \mathsf{Model}'(\text{PROMPT}, \mathsf{t}_1, \ldots, \mathsf{t}_{i-1})$
  If $\mathsf{p}_i(0) \geq 1/2$, sample $\mathsf{t}_i \sim \text{Ber}(2r_i\mathsf{p}_i(1))$
  Else, sample $\mathsf{t}_i \sim \text{Ber}(1 - 2(1 - r_i)\mathsf{p}_i(0))$
Output $\mathsf{t} = (\mathsf{t}_1, \ldots, \mathsf{t}_i)$

$\underline{\mathsf{Recover}(\mathsf{t})\text{:}}$
Output t.

[a] We abuse notation a bit, as the token alphabet is $\{0, 1\}$ and does not contain the `done` token; more precisely we stop and output the response when we see the binary encoding of the `done` token.

Somewhat amazingly, $\mathsf{Recover}$ is simply the identity function. To see why $\mathsf{Generate}$ is equivalent to $\overline{\mathsf{Model}}'$, let us observe that if $r_i \sim \text{Ber}(1/2)$, t_i is sampled from p_i:

$$\text{If } \mathsf{p}_i(0) \geq 1/2 : \mathop{\mathbb{E}}_{r_i}[\mathsf{t}_i] = \frac{1}{2} \cdot 0 + \frac{1}{2} \cdot 2 \cdot \mathsf{p}_i(1) = \mathsf{p}_i(1)$$
$$\text{Else} : \mathop{\mathbb{E}}_{r_i}[\mathsf{t}_i] = \frac{1}{2}(1 - 2\mathsf{p}_i(0)) + \frac{1}{2}(1) = 1 - \mathsf{p}_i(0) = \mathsf{p}_i(1)$$

Therefore, in either case $\mathsf{t}_i \sim \text{Ber}(\mathsf{p}_i(1))$ as desired.

The likelihood that $\mathsf{t}_i = r_i$ depends on the entropy present in the next-token distribution p_i. At one extreme, if p_i is entirely deterministic, $\mathsf{t}_i = r_i$ with probability 1/2. On the other hand, if $\mathsf{p}_i(0) = \mathsf{p}_i(1) = 1/2$, we have that $\mathsf{t}_i = r_i$ with probability 1. Christ and Gunn quantify this tradeoff by proving a bound on the error rate of recovery in terms of the entropy of the response [CG24, Lemma 21]. Roughly, they show that if the average entropy of the p_i's is some constant ε, there is another constant ε' such that the error rate of $\mathsf{Recover}$ is at most $(1/2 - \varepsilon')$ with overwhelming probability.

The binary transformation is very clean in theory, allowing works following [CGZ24] to consider only binary language models. Furthermore, the fact that $\mathsf{Recover}$ is simply the identity function makes robustness very easy to analyze: changing a bit of t changes that bit of $\mathsf{Recover}(\mathsf{t})$.

However, the binary transformation causes the error rate of $\mathsf{Recover}$ to be quite high in terms of concrete parameters. Typical LLM-generated text has, generously, about one bit of entropy per token. When converted to binary, there are roughly $1/\log|\mathcal{T}|$ bits of entropy per response bit. In practice, models have alphabets $\mathcal{T}$ ranging from tens of thousands to hundreds of thousands of tokens. This means that many bits t_i are essentially deterministic.

A natural alternative that mitigates this problem is working with the original $\overline{\mathsf{Model}}$, over its larger alphabet $\mathcal{T}$, and embedding one PRC bit per token. Let $h : \mathcal{T} \to \{0, 1\}$ be a balanced function, and consider the following $(\mathsf{Generate}, \mathsf{Recover})$:

A simple hash-based $(\mathsf{Generate}, \mathsf{Recover})$ **for arbitrary** $\mathcal{T}$

$\underline{\mathsf{Generate}(\text{PROMPT}; r)\text{:}}$
While $\mathsf{t}_i \neq \mathtt{done}$:
  $i = i + 1$
  Let $\mathsf{p}_i \leftarrow \mathsf{Model}(\text{PROMPT}, \mathsf{t}_1, \ldots, \mathsf{t}_{i-1})$
  Sample $\mathsf{t}_i \sim \mathsf{p}_i$
  If $h(\mathsf{t}_i) = r_i$, continue
  Else, resample $\mathsf{t}_i \sim \mathsf{p}_i$
Output $\mathsf{t} = (\mathsf{t}_1, \ldots, \mathsf{t}_i)$

$\underline{\mathsf{Recover}(\mathsf{t})\text{:}}$
Output $h(\mathsf{t}_1), \ldots, h(\mathsf{t}_{\text{len}\,\mathsf{t}})$.

Those familiar with steganography may recognize this rejection sampling strategy from an early scheme of Hopper, Langford, and von Ahn [HLVA02].

This scheme maintains much of the niceness of the binary transformation. The $\mathsf{Recover}$ procedure is still quite simple, and changing one token translates to changing at most one PRC bit. Now, the error rate of $\mathsf{Recover}$ depends on the amount of entropy in $h(\mathsf{t}_i)$ for $\mathsf{t}_i \sim \mathsf{p}_i$.

A slight downside is that for this choice of $\mathsf{Recover}$, we can no longer prove a relationship between the amount of entropy in the response and the error rate of $\mathsf{Recover}$. Consider a response where every p_i is uniformly distributed over $h^{-1}(0)$. This will have high entropy—far higher than is present in responses to typical prompts—yet the error rate of $\mathsf{Recover}$ will be 1/2.

One might attempt to prevent this issue by choosing h at random and keeping it secret. Unfortunately, now consider a response where every p_i is uniform over the same two tokens. With probability 1/2, these tokens have the same output under h, and the response has high entropy yet maximal recovery error.

Fortunately, Mazor et al. [MMP26] show that if we partition the response into blocks and choose a *different, random* h_i for each block, one can finally obtain a provable relationship between response entropy and recovery error ([MMP26, Claim 2]). Their scheme implicitly uses the following pair $(\mathsf{Generate}, \mathsf{Recover})$, for block length ℓ and random $h_0, \ldots, h_k$ generated along with the secret key:

Multi-hash-based $(\mathsf{Generate}, \mathsf{Recover})$ [MMP26]

$\underline{\mathsf{Generate}(\text{PROMPT}; r)}$:
$i = 0, j = 0, \mathsf{t}_0 = \perp$
Until last output token is `done`:
  For $j \in [\ell]$:
    Let $\mathsf{p}_{i\ell+j} \leftarrow \mathsf{Model}(\text{PROMPT}, \mathsf{t}_1, \ldots, \mathsf{t}_{i\ell+j-1})$
    Sample $\mathsf{t}_{i\ell+j} \sim \mathsf{p}_{i\ell+j}$
    If $h_i(\mathsf{t}_{i\ell+j}) = r_{i\ell+j}$, continue
    Else, resample $\mathsf{t}_{i\ell+j} \sim \mathsf{p}_{i\ell+j}$
  $i = i + 1$
Output $\mathsf{t} = (\mathsf{t}_1, \ldots, \mathsf{t}_{i\ell+j})$

$\underline{\mathsf{Recover}(\mathsf{t})}$:
For $j \in [\text{len}\, \mathsf{t}]$:
  $i = \lfloor j/\ell \rfloor$
  $s_j = h_i(\mathsf{t}_j)$
Output s

On robustness. For all of the watermarks we have seen so far, robustness of the PRC corresponds neatly to robustness of the watermark. Each token maps to one PRC bit, so a rate p of token substitutions results in a rate of at most p PRC substitutions.[17] One can view the error added by $\mathsf{Recover}$ as an *embedding channel* which makes substitution errors at a rate dependent on the amount of entropy in the response. A watermark is robust to an adversary changing a fraction δ of tokens if the PRC is robust to $\varepsilon + \delta$ substitutions, where ε is the rate of embedding error.

Furthermore, the power of the watermark adversary maps exactly to the power of the PRC robustness adversary. If we want watermark robustness to an adversary that changes tokens obliviously, a PRC with oblivious robustness suffices. If the adversary has query access to Wat, adaptive robustness suffices. If the adversary has query access to Wat and Detect, adaptive-with-decoder robustness (and therefore ideal security) suffices. If the watermark adversary is full-information (e.g., knows the detection key), we need worst-case robustness

[17] We say "at most" because for the hash-based schemes, a token t_i may be changed to t'_i such that $h(\mathsf{t}_i) = h(\mathsf{t}'_i)$, resulting in no error.

of the PRC.

One can imagine practical applications motivating robustness against these various adversaries. For example, if we want anyone to be able to detect the watermark, we can publish the detection key; if we still want robustness we should use a PRC with worst-case robustness. Of course, if we publish the detection key the watermark is no longer undetectable, as Detect trivially serves as a distinguisher. However, undetectability served primarily to preserve quality of the content, which is not affected by publishing the detection key.

On the PRC alphabet. Recall that an important requirement of pseudorandom codes, and one that makes them difficult to construct, is that the alphabet is *binary.*[18] To see where this requirement comes from, observe that all of the pairs (Generate, Recover) that we just saw embed one PRC symbol in each token. For an alphabet Σ, constant probability of recovery requires that the token has $\Omega(\log|\Sigma|)$ entropy. In typical LLM-generated text, tokens have on the order of one bit of entropy, necessitating a constant-sized alphabet.

To increase the available entropy, one could instead embed each PRC symbol in a chunk of k tokens. However, this results in an unfortunate tradeoff with robustness: if any of the k tokens change, the PRC symbol is lost. If we wish to use $|\Sigma| = \omega(1)$, we need $k = \omega(1)$, and we can remove the watermark by making a rate $1/o(1)$ of substitutions, changing one token in each chunk. Therefore, if we want robustness to a constant rate of token substitutions, this chunking method fails.

7.1.3 Enhanced PRC watermarks

So far, we have used only zero-bit PRCs to construct watermarks. We will now see that multi-bit PRCs can be used to build watermarks with extra properties.

First, we will use a signature scheme Sig = (Gen, Sign, Vrfy) to construct a watermark that is *unforgeable* by an adversary that does not know the secret key. The scheme involves signing part of the response, then using this signature as the message of a PRC codeword used as r in generating the remainder of the response. This is a simplified variant of a scheme by Christ and Gunn [CG24], which is unforgeable even by an attacker that knows the detection key.

[18] Or, equivalently, constant-sized, as one can convert codewords to binary and incur a constant loss in the error tolerance.

Unforgeable watermark [CG24]

$\underline{\mathsf{Setup}(1^\lambda)}$:
Let $\mathsf{PRC.sk} \leftarrow \mathsf{PRC.KeyGen}(1^\lambda)$
Let $(\mathsf{Sig.pk}, \mathsf{Sig.sk}) \leftarrow \mathsf{Sig.Gen}(1^\lambda)$
Output $\mathsf{sk} = (\mathsf{PRC.sk}, \mathsf{Sig.sk}, \mathsf{Sig.pk})$

$\underline{\mathsf{Wat}_{\mathsf{sk}}(\text{PROMPT})}$:
Generate a partial response $c_1 \leftarrow \mathsf{Generate}(\text{PROMPT}; \mathcal{U})$
Compute $\sigma \leftarrow \mathsf{Sig.Sign}_{\mathsf{Sig.sk}}(c_1)$
Let $r \leftarrow \mathsf{PRC.Encode}(\mathsf{PRC.sk}, \sigma)$
Generate $c_2 \leftarrow \mathsf{Generate}(\text{PROMPT}||c_1; r)$
Output $c_1||c_2$

$\underline{\mathsf{Detect}_{\mathsf{sk}}(c)}$:
Parse c as $c_1||c_2$
Let $r' \leftarrow \mathsf{Recover}(c_2)$
Let $\sigma' \leftarrow \mathsf{PRC.Decode}(\mathsf{PRC.sk}, r')$
If $\sigma' = \perp$ or $\mathsf{Sig.Vrfy}_{\mathsf{Sig.pk}}(c_1, \sigma') = 0$, output false
Else, output true and c_1

Specifically, this scheme is unforgeable in that if the watermark outputs true, the corresponding c_1 must have been generated by the watermarked model.

We will now use the PRC's ability to encode longer messages to construct a *recoverable* watermark. For any sufficiently close modification c' of a piece of content $c \leftarrow \mathsf{Wat}$, the detector not only identifies the watermark but also recovers the prompt and *unmodified* content c output by Wat.

Recoverable watermarks are useful in many applications, and a notion of recoverability is defined formally and studied in [LSS26]. For example, suppose a student wrote an essay using an LLM but claims that the LLM was only used to reword their completed essay. A standard watermark would identify the essay as being written with the help of an LLM. Using a recoverable watermark, in particular its ability to recover the prompt, we could determine whether the student asked the LLM to reword a full essay, or whether they asked the LLM to write an essay from scratch.

As another example, suppose an adversary generates a lovely paragraph using a watermarked LLM, changes a few words to make it horribly offensive, then posts it online. A standard robust watermark would indicate simply that this offensive paragraph was generated with the help of the watermarked LLM; however, it would not tell us whether the LLM was responsible for the offensive part. With a recoverable watermark, we could see that the watermarked LLM output a benign paragraph and the offensive portion was added later.

Our construction of a recoverable watermark will involve embedding all information necessary for recovery in the PRC message. This will include both the prompt and any randomness used in $\mathsf{Generate}$ and Encode. Recall that $\mathsf{Generate}(\text{PROMPT}; r)$ may use some additional randomness beyond r. Here, we use x to denote this additional randomness and write $\mathsf{Generate}(\text{PROMPT}; r; x)$ to mean the $\mathsf{Generate}$ algorithm that is deterministic once x is

fixed. We will use similar notation for Encode.

We will sample both PRF and PRC secret keys $\mathsf{sk} = (\mathsf{PRC.sk}, \mathsf{PRF.sk})$. For ease of notation, we will write $\mathsf{PRC.Encode}(\mathsf{sk}, ...)$ to mean $\mathsf{PRC.Encode}(\mathsf{PRC.sk}, ...)$, and similarly for $\mathsf{PRF.sk}$.

Recoverable watermark

$\underline{\mathsf{Setup}(1^\lambda)}$:
Let $\mathsf{PRC.sk} \leftarrow \mathsf{PRC.KeyGen}(1^\lambda)$ and $\mathsf{PRF.sk} \leftarrow \{0,1\}^\lambda$
Output $\mathsf{sk} = (\mathsf{PRC.sk}, \mathsf{PRF.sk})$

$\underline{\mathsf{Wat}_{\mathsf{sk}}(\text{PROMPT})}$:
$s \leftarrow \{0,1\}^\lambda$, $x \leftarrow \mathsf{PRF}_{\mathsf{sk}}(s||1)$, $y \leftarrow \mathsf{PRF}_{\mathsf{sk}}(s||2)$
$r \leftarrow \mathsf{PRC.Encode}(\mathsf{sk}, (\text{PROMPT}, s); y)$
Output $\mathsf{Generate}(\text{PROMPT}; r; x)$

$\underline{\mathsf{Detect}_{\mathsf{sk}}(c)}$:
Let $r' \leftarrow \mathsf{Recover}(c)$ and $z \leftarrow \mathsf{PRC.Decode}(\mathsf{sk}, r')$
If $z = \perp$, output false
Otherwise, parse z as (PROMPT', s')
Let $x' \leftarrow \mathsf{PRF}_{\mathsf{sk}}(s'||1)$ and $y' \leftarrow \mathsf{PRF}_{\mathsf{sk}}(s'||2)$
Let $r \leftarrow \mathsf{PRC.Encode}(\mathsf{sk}, (\text{PROMPT}', s'); y')$
Output true, PROMPT', and $\mathsf{Generate}(\text{PROMPT}'; r; x')$

7.1.4 Non-PRC watermarks

Other LLM watermarks that do not use PRCs (and therefore are not undetectable or are not robust) can also be viewed using $(\mathsf{Generate}, \mathsf{Recover})$.

For example, Kuditipudi et al. [KTHL24] construct an LLM watermark that is *distortion-free*, meaning that a single response from the watermarked model is distributed identically to a single response from the original model. One can view this scheme as *fixing* the randomness r used by $\mathsf{Generate}$, storing it as the secret key. The first response will be distributed as $\mathsf{Generate}(\text{PROMPT}; \mathcal{U})$ which is the original model. To detect, one runs $\mathsf{Recover}$ and checks if the resulting string is close to r in edit distance, giving edit robustness.

Recall that $\mathsf{Generate}$ may use additional randomness beyond r, as is the case in [CG24]. So while fixing r reduces the diversity of responses, it does not necessarily make the model entirely deterministic.

Distortion-free watermark for $(\mathsf{Generate}, \mathsf{Recover})$ [KTHL24, simplified]

$\underline{\mathsf{Setup}(1^\lambda)}$:
Output $\mathsf{sk} = r \leftarrow \{0,1\}^n$

$\underline{\mathsf{Wat}_{\mathsf{sk}}(\text{PROMPT})}$:
Output $\mathsf{Generate}(\text{PROMPT}; r)$

$\underline{\mathsf{Detect}_{\mathsf{sk}}(c)}$:
Let $r' \leftarrow \mathsf{Recover}(c)$
Output true iff $d_E(r, r') \ll n/2$

Another relevant scheme is that of Christ, Gunn, and Zamir [CGZ24], which is undetectable but not robust. This scheme can be viewed as sampling a partial response c_1 from the original model, then deriving r from a PRF applied to c_1, and running $c_2 \leftarrow \mathsf{Generate}(\text{PROMPT}||c_1; r)$ to obtain the full response $(c_1||c_2)$. Pseudorandomness of the PRF implies undetectability, provided that we never reuse the same partial response c_1 as input to the PRF. A key observation of [CGZ24] is that one can use the probability distributions p output by Model to compute a notion of "empirical entropy" of c_1 given the prompt. If we continue appending tokens to c_1 until it has at least λ empirical entropy, we can ensure that we never reuse the same partial response as input to the PRF.

To detect the watermark, we rederive r by applying the PRF to c_1, then check similarity between r and $\mathsf{Recover}(c_2)$.[19] Note that we must search over the ending index of c_1, which may vary.

This scheme lacks robustness because changing any token in c_1 prevents us from recovering r; the PRF applied to a changed c_1' will be arbitrarily far from r.

[19] [CGZ24] uses a slightly different way to check similarity, using a statistical test, but the scheme below captures theirs in spirit.

Undetectable watermark for (Generate, Recover) [CGZ24]

$\underline{\mathsf{Setup}(1^\lambda)}$:
Output $\mathsf{sk} \leftarrow \{0,1\}^n$

$\underline{\mathsf{Wat}_{\mathsf{sk}}(\text{PROMPT})}$:
While the empirical entropy of c_1 given PROMPT is $< \lambda$:
    Sample a single token $\mathsf{t} \leftarrow \mathsf{Generate}(\text{PROMPT}||c_1; \mathcal{U})$
    Append t to c_1
Let $r \leftarrow \mathsf{PRF}_{\mathsf{sk}}(c_1)$
Let $c_2 \leftarrow \mathsf{Generate}(\text{PROMPT}||c_1; r)$
Output $c_1||c_2$

$\underline{\mathsf{Detect}_{\mathsf{sk}}(c)}$:
For $i \leq \operatorname{len} c$:
    Let $c_1 = c[1:i]$ and $c_2 = c[i+1 : \operatorname{len} c]$
    Let $r \leftarrow \mathsf{PRF}_{\mathsf{sk}}(c_1)$ and $r' \leftarrow \mathsf{Recover}(c_2)$
    If $d_H(r, r') \ll (\operatorname{len} c_2)/2$, output true
Output false

7.2 Image watermarks

Gunn, Zhao, and Song [GZS25] use pseudorandom codes to construct an undetectable watermark for *latent diffusion models* [RBL+22]. A latent diffusion model is a generative algorithm A that takes as input a prompt PROMPT and outputs an image. It generates an image by first sampling a Gaussian vector $z \in \mathbb{R}^n$ that represents the image's higher-order features, then uses the prompt to transform z into an image. A bit more precisely, it uses a pair of algorithms called an *encoder* E and a *decoder* D, along with a de-noising procedure f that can be approximately inverted using a process Inv. On input PROMPT, it does the following:

1. Sample $z \sim \mathcal{N}(0, I_n)$
2. Let $z' \leftarrow f(\text{PROMPT}, z)$
3. Output $c \leftarrow D(z') \in \mathbb{R}^d$

Latent diffusion models are trained so that applying the encoder E to an image c approximately recovers the denoised latent vector z'. Applying Inv to $E(c)$ then yields a vector that is close to z. This closeness appears to hold even for images c' that are semantically similar to c.

Gunn et al. show that latent diffusion models are naturally randomness-recoverable algorithms:

(Generate, Recover) **for latent diffusion models** [GZS25]

Generate(PROMPT; r):
Sample $z \sim \mathcal{N}(0, I_n)$ conditioned on $\mathsf{Sign}(z_i) = (-1)^{r_i}$ for all $i \in [n]$
Output $c \leftarrow D(f(\text{PROMPT}, z))$

Recover(c):
Output the vector of signs of $\mathsf{Inv}(E(c))$.

Note that both Generate and Recover use learned algorithms that are difficult to analyze. While the autoregressive structure of LLMs allowed us to prove that robustness holds so long as the response has sufficient entropy, here we lack such structure. Although Gunn et al. do not prove robustness or completeness of their scheme, they do show empirically that their watermark persists under a range of practical attacks.

A particular concern is how changes to c influence the recovery error, due to the use of E. An adversary could take an image c generated using denoised latents z', and craft a similar image c' such that $E(c')$ is very far from z'.[20] Gunn et al. measure the success of one such attack empirically ([GZS25, Figure 8]). However, provable robustness of the watermark would require a provable bound on the adversarial robustness of E, which is not known for latent diffusion models.

Despite the lack of provable robustness, this watermark does achieve a nice *kind* of robustness. Since the watermark is embedded in the latent *features*, it should be detectable even if the image is modified, as long as these features remain similar. These features are not easy to interpret, as they are learned, but one can think of them as representing properties like how bright an image is, or whether it is a landscape. This means that the watermark could be detected even if *every pixel is changed*, as long as these more global features are preserved. This is in contrast to the LLM watermarks we just saw, where changing every word of the text, even to close synonyms, will remove the watermark.[21]

Another important difference between this watermark and LLM watermarks is the usage of model-specific information in Recover. Previously, Recover did not depend on the underlying model. In contrast, here the encoder and decoder (E, D) are trained *together*; if c is generated using D and latents z', the encoder E' of another model may recover latents that are arbitrarily far from z'. Therefore, at detection time one must at least suspect which model was used to generate the given image.

This also makes detection more computationally intensive. If the model is proprietary, enabling detection by the general public also becomes problematic as one might not want to release E.

Similarly to the PRC-based watermark for LLMs, this image watermark has a distortion-free analogue obtained by using a fixed r included in the secret key. This roughly corresponds to a popular and practical image watermark, Gaussian Shading [YZC+24]. A more recent work combines Gaussian Shading with pseudorandom codes [YZC+25].

[20] This is called an adversarial example, a well-studied concept in machine learning. Adversarial examples abound and are easy to find in most practical neural networks.

[21] Constructing LLM watermarks with similar semantic robustness is a very interesting open problem (see Section 8).

7.3 Video watermarks

Pseudorandom codes have recently been used to watermark outputs of video models [HLLL25, ZZJ+26]. We refer the reader to those papers for further details.

7.4 Robust steganography

Though we do not go into detail here, pseudorandom codes are also useful for *robust steganography*. Steganography allows a sender to craft ciphertexts that "look like regular content" from some distribution. That is, let $\mathcal{C}$ be some distribution of naturally occurring messages (e.g., text messages). Steganography is essentially an encryption scheme where, for any fixed message m, the distribution of $\mathsf{Encrypt}(\mathsf{sk}, \mathsf{m})$ is computationally indistinguishable from $\mathcal{C}$. A steganography scheme is robust if decryption succeeds even when the ciphertext undergoes a constant rate of substitution errors.

Like in our failed attempt to construct PRCs generically (Section 3), observe that applying an error-correcting code to a steganographic ciphertext could alter it in a distinguishable way. An easy example is a repetition code; even if ciphertexts look like benign text messages, encoding them with a repetition code will be obvious.

Christ and Gunn [CG24, Section 8] showed that pseudorandom codes allow us to construct the first robust stateless steganography scheme.

8 Open questions

Here, we highlight some of the most interesting open questions surrounding pseudorandom codes and their applications.

So far, the only language model watermarks that we have considered are embedded at the token-level. That is, changing every word of the response will remove the watermark, even if we replace each word with a very close synonym. A nicer and more challenging form of robustness is *semantic robustness*: the watermark should remain after any modifications that preserve the text's semantics, such as paraphrasing. The PRC-based image watermark of Gunn et al. that we saw earlier can be thought of as semantically robust. The watermark is embedded in the features of the image, so even transformations that change every pixel will preserve the watermark so long as the image's high-level features remain.

Therefore, it seems entirely possible to construct a similar watermark for language models. The following question would be very interesting to answer, especially if the watermark is practical (in particular, requires a modest amount of text to detect):

Open Question 1 (Application)
Can we use pseudorandom codes to construct an undetectable and *semantically robust* watermark for language models?

Unfortunately, current PRC-based LLM watermarks require a prohibitive amount of text to detect, on the order of thousands of tokens. This is due to the concrete tradeoff between the

error rate and codeword length of practical constructions. Essentially, for the constructions that involve planting a t-sparse structure, robustness relies on this structure being untouched by error, which happens with probability $(1-\varepsilon)^t$ for error rate ε. Therefore, we actually must plant $(1-\varepsilon)^{-t}$ of these structures for the expected number of error-free structures to be one. In the constructions we have, the length of the codeword grows at least linearly with the number of structures we plant. For text, the error introduced just by Recover is quite high, at around 40%. Therefore, for concrete parameters, the codeword length becomes quite long.

Open Question 2 (Construction)
Can we construct pseudorandom codes with a better concrete tradeoff between robustness and codeword length?

As mentioned, two recent works [GGW25, DMR25] established that pseudorandom codes are black-box separated from a wide range of cryptographic primitives. While these works say that in some sense pseudorandom codes are more powerful than other primitives, they have not found many applications in cryptography beyond watermarks and steganography. One exception is a recent paper by Lu, Silbak, and Wichs, which uses adaptively robust pseudorandom codes to construct seeded error-correcting codes achieving Shannon capacity against a computationally bounded channel [LSW26].

Very recent PRC constructions have also yielded new public-key encryption schemes, but these are products of the constructions rather than implied generically by PRCs [CMY26, DFV26, DJK+26]. It would be very interesting to show that pseudorandom codes combined with an assumption A imply public-key encryption, for a choice of A that is not known to imply public-key encryption when combined with one-way functions. It seems unlikely that PRCs alone imply public-key encryption, and showing this impossibility would also be interesting.

Another direction for applying PRCs to cryptography is in doubly-efficient private information retrieval (DEPIR). Interestingly, the black-box separation of Garg et al. [GGW25] uses the crypto-oracles that were introduced in a black-box separation for DEPIR [LMW25]. Also, a new construction of pseudorandom codes [CGG+25] relies on an assumption introduced to construct DEPIR. An explicit connection between DEPIR and pseudorandom codes would be very interesting to show, though a generic construction of DEPIR from PRCs alone would be surprising.

Open Question 3 (Application)
Can we show purely cryptographic applications of pseudorandom codes, perhaps connections to DEPIR or public-key encryption?

Open Question 4 (Theory)

It has already been shown that there is no black-box construction of pseudorandom codes from public-key encryption [GGW25]. Can we similarly show that there is no black-box construction of public-key encryption from pseudorandom codes?

Similarly, it would be interesting to show applications of ideal PRCs to other areas of theoretical computer science. An ideal PRC is essentially an efficient way to implement a random code with a superpolynomially large codebook. Perhaps this is useful in areas beyond cryptography, though this question is a bit vague.

Open Question 5 (Application)

Can we show applications of ideal PRCs to areas of theoretical computer science beyond cryptography?

Recall that [AAC+25] shows only that the specific LDPC-based PRC is adaptively robust. It would be very interesting to have a generic transformation that converts *any* obliviously robust PRC to an adaptively robust PRC, even incurring a constant loss to the error tolerance.

Open Question 6 (Generic transformation)

Can we show a fully generic transformation from oblivious robustness to adaptive robustness?

A direction left open by Alrabiah et al. [AAC+25] is constructing a zero-bit ideal pseudorandom code robust to a high rate $p \in (1/4, 1/2)$ of substitutions. While Alrabiah et al. show that the LDPC-based PRC is adaptively robust to any constant rate of errors up to $1/2$, their ideal PRC is limited to robustness $< 1/4$. This was included as an open question in the previous version of this survey, and it has since been solved by George Lu and Jad Silbak.

~~**Open** Question 7 (Construction or generic transformation)~~

Can we construct a zero-bit PRC with p-robust ideal security, for some $p \in (1/4, 1/2)$?

Solved by George Lu and Jad Silbak, who kindly provided a proof (Section 8.1).

In particular, they show a generic transformation from any *adaptively* $(p+\varepsilon)$-robust (for any constant $\varepsilon > 0$) zero-bit PRC to an p-ideal one. This transformation applies to both secret-key and public-key PRCs.[22] Since [AAC+25] showed we have adaptively p-robust public-key PRCs for p up to $1/2$, this transformation yields public-key zero-bit PRCs for *any* $p \in (1/4, 1/2)$.

[22]For public-key PRCs, a slightly weaker notion than adaptive robustness suffices.

The proof (given in Section 8.1) uses techniques from their work on computationally bounded error-correction [LSW26].

Suppose we have a PRC that is robust to a rate ε of substitutions. Can we generically convert this into a PRC that is robust to an $\varepsilon' > \varepsilon$ error rate, or show that this is impossible? This would be interesting for any form of substitution robustness (e.g., random, oblivious, adaptive).

Open Question 8 (Generic transformation)
Is there a way to generically boost the *rate* of substitution robustness, or is there a black-box separation?

It remains open to convert a binary substitution-robust PRC into a binary edit-robust PRC, maintaining constant error tolerance. Currently, the best construction is that of Golowich and Moitra [GM24], which tolerates an $O(1/\log n)$ rate of edits.

Open Question 9 (Generic transformation)
We know that PRCs robust to random substitutions imply PRCs robust to oblivious substitutions. Is the same true for edits?

Finally, the challenge of constructing PRCs has led to the introduction of new assumptions. It would be very interesting to show that any of these assumptions are false.

Open Question 10 (Attack)
Can we show attacks against any of the new assumptions introduced for recent constructions?

First, assumptions that may be more difficult to attack are (subexponential) LPN, log-local weak PRFs, and security of Goldreich's PRG. These problems have withstood decades of study, and significant attacks against log-local weak PRFs would earn you prize money from Avrim Blum [Blu03].

Newer assumptions include:

- Permuted codes [CGG$^+$25] and the related permuted puzzles conjecture [BIPW17, CHR17, BW21, BHMW21]
- Noisy McEliece [DJK$^+$26]
- Distorted McEliece [DFV26]
- Dense-Planted LPN [CMY26]

8.1 Solution to Question 7

The following proof was written by George Lu and Jad Silbak and is included with their permission.

We construct optimally (constant p arbitrarily close to $\frac{1}{2}$) robust zero-bit ideal pseudorandom codes. This adapts the PRF sharpening trick of [AAC+25] to large domain (non-ideal) list-decodable PRCs from [LSW26], which follows generically from (non-ideal) zero-bit PRCs of the same robustness.

We use δ to denote relative Hamming distance, and $\xleftarrow{R} \mathcal{D}$ to denote uniform random sampling from $\mathcal{D}$.

Construction 2 (Zero-Bit Ideal PRC)**.** Our construction makes use of the following. Take $0 < p < \frac{1}{2}$ to be an arbitrary constant.

- A secret-key list-decodable pseudorandom code $\Pi_{\mathsf{list}} = (\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ with robustness p, message space $\{0,1\}^{2\lambda}$, randomness space $\{0,1\}^{r(\lambda)}$, and codeword space $\{0,1\}^{t(\lambda)}$ for some $r(\lambda), t(\lambda) \in \mathsf{poly}(\lambda)(\lambda)$.
- A pseudorandom function $\Pi_{\mathsf{PRF}} = (\mathsf{Setup}, \mathsf{Eval})$ with domain $\{0,1\}^{\lambda}$ and range $\{0,1\}^{\lambda + r(\lambda)}$.

The construction of an ideal PRC is as follows:

- $\mathsf{KeyGen}(1^{\lambda})$: On input security parameter λ, the key generation algorithm samples
$$\mathsf{sk}_{\mathsf{list}} \leftarrow \Pi_{\mathsf{list}}.\mathsf{KeyGen}(1^{\lambda}) \quad , \quad \mathsf{sk}_{\mathsf{PRF}} \leftarrow \Pi_{\mathsf{PRF}}.\mathsf{Setup}(1^{\lambda}),$$
and outputs $\mathsf{sk} = (\mathsf{sk}_{\mathsf{list}}, \mathsf{sk}_{\mathsf{PRF}})$.
- $\mathsf{Encode}(\mathsf{sk})$: On input secret key $\mathsf{sk} = (\mathsf{sk}_{\mathsf{list}}, \mathsf{sk}_{\mathsf{PRF}})$, the encoding algorithm samples $z \xleftarrow{R} \{0,1\}^{\lambda}$ and evaluates $(R_1, R_2) \leftarrow \Pi_{\mathsf{PRF}}.\mathsf{Eval}(\mathsf{sk}_{\mathsf{PRF}}, z)$, where $R_1 \in \{0,1\}^{\lambda}$ and $R_2 \in \{0,1\}^{r(\lambda)}$. It computes $c_z \leftarrow \Pi_{\mathsf{list}}.\mathsf{Encode}(\mathsf{sk}_{\mathsf{list}}, (z, R_1); R_2)$ and outputs c_z.
- $\mathsf{Decode}(\mathsf{sk}, c')$: On input secret key $\mathsf{sk} = (\mathsf{sk}_{\mathsf{list}}, \mathsf{sk}_{\mathsf{PRF}})$ and candidate codeword c', the decoding algorithm computes $T \leftarrow \Pi_{\mathsf{list}}.\mathsf{Decode}(\mathsf{sk}_{\mathsf{list}}, c')$. For each $(z, R_1^*) \in T$, it evaluates
$$(R_1, R_2) \leftarrow \Pi_{\mathsf{PRF}}.\mathsf{Eval}(\mathsf{sk}_{\mathsf{PRF}}, z)$$
and checks whether
$$R_1^* = R_1 \quad \wedge \quad \delta(\Pi_{\mathsf{list}}.\mathsf{Encode}(\mathsf{sk}_{\mathsf{list}}, (z, R_1); R_2), c') \leq p.$$
It returns 1 if any candidate satisfies both checks, and $\perp$ otherwise.

Theorem 12. *construction 2 is an ideal p-robust zero-bit PRC.*

Proof. We consider the following sequence of hybrids:

- $\mathsf{Hyb}_{\mathsf{real}}$: This is the real security experiment, instantiated with the above construction.
  1. The challenger samples
  $$\mathsf{sk}_{\mathsf{list}} \leftarrow \Pi_{\mathsf{list}}.\mathsf{KeyGen}(1^{\lambda}) \quad , \quad \mathsf{sk}_{\mathsf{PRF}} \leftarrow \Pi_{\mathsf{PRF}}.\mathsf{Setup}(1^{\lambda}).$$

2. As many times as desired, the adversary submits the following:
   (a) On an encoding query, the challenger samples $z \xleftarrow{R} \{0,1\}^\lambda$ and evaluates
   $$(R_1, R_2) \leftarrow \Pi_{\mathsf{PRF}}.\mathsf{Eval}(\mathsf{sk}_{\mathsf{PRF}}, z).$$
   It computes
   $$c_z \leftarrow \Pi_{\mathsf{list}}.\mathsf{Encode}(\mathsf{sk}_{\mathsf{list}}, (z, R_1); R_2)$$
   and returns c_z to the adversary.
   (b) On a decoding query c', the challenger computes $T \leftarrow \Pi_{\mathsf{list}}.\mathsf{Decode}(\mathsf{sk}_{\mathsf{list}}, c')$. For each $(z, R_1^*) \in T$, it computes $(R_1, R_2) \leftarrow \Pi_{\mathsf{PRF}}.\mathsf{Eval}(\mathsf{sk}_{\mathsf{PRF}}, z)$ and checks whether
   $$R_1^* = R_1 \quad \wedge \quad \delta(\Pi_{\mathsf{list}}.\mathsf{Encode}(\mathsf{sk}_{\mathsf{list}}, (z, R_1); R_2), c') \leq p.$$
   It returns 1 if any candidate satisfies both checks, and $\perp$ otherwise.
3. The adversary returns a bit $\hat{b}$, the output of the experiment.

- Hyb_1 : In this experiment, we replace PRF invocations with a (lazily sampled) true random function.

1. The challenger initializes
   $$\mathsf{sk}_{\mathsf{list}} \leftarrow \Pi_{\mathsf{list}}.\mathsf{KeyGen}(1^\lambda) \quad , \quad \mathsf{transcript}_{\mathsf{PRF}} = \varnothing.$$
2. As many times as desired, the adversary submits the following:
   (a) On an encoding query, the challenger samples $z \xleftarrow{R} \{0,1\}^\lambda$. If $\mathsf{transcript}_{\mathsf{PRF}}$ contains no entry with first component z, it samples $(R_{1,z}, R_{2,z}) \xleftarrow{R} \{0,1\}^\lambda \times \{0,1\}^{r(\lambda)}$ and sets
   $$\mathsf{transcript}_{\mathsf{PRF}} := \mathsf{transcript}_{\mathsf{PRF}} \cup \{(z, (R_{1,z}, R_{2,z}))\}.$$
   Otherwise, it retrieves the stored pair $(R_{1,z}, R_{2,z})$. In either case, it computes
   $$c_z \leftarrow \Pi_{\mathsf{list}}.\mathsf{Encode}(\mathsf{sk}_{\mathsf{list}}, (z, R_{1,z}); R_{2,z})$$
   and returns c_z to the adversary.
   (b) On a decoding query c', the challenger computes $T \leftarrow \Pi_{\mathsf{list}}.\mathsf{Decode}(\mathsf{sk}_{\mathsf{list}}, c')$. For each $(z, R_1^*) \in T$, if $\mathsf{transcript}_{\mathsf{PRF}}$ contains no entry with first component z, it samples $(R_{1,z}, R_{2,z}) \xleftarrow{R} \{0,1\}^\lambda \times \{0,1\}^{r(\lambda)}$ and sets
   $$\mathsf{transcript}_{\mathsf{PRF}} := \mathsf{transcript}_{\mathsf{PRF}} \cup \{(z, (R_{1,z}, R_{2,z}))\}.$$
   Otherwise, it retrieves the stored pair $(R_{1,z}, R_{2,z})$. It checks whether any $(z, R_1^*) \in T$ satisfies
   $$R_1^* = R_{1,z} \quad \wedge \quad \delta(\Pi_{\mathsf{list}}.\mathsf{Encode}(\mathsf{sk}_{\mathsf{list}}, (z, R_{1,z}); R_{2,z}), c') \leq p,$$
   returning 1 if so and $\perp$ otherwise.

3. The adversary returns a bit $\hat{b}$, the output of the experiment.

- Hyb_2 : In this experiment, decoding queries return 1 only when a previous encoding query produced a codeword within relative Hamming distance p of c'.

  1. The challenger initializes

  $$\mathsf{sk}_{\mathsf{list}} \leftarrow \Pi_{\mathsf{list}}.\mathsf{KeyGen}(1^\lambda) \quad , \quad \mathsf{transcript}_{\mathsf{PRF}} = \varnothing \quad , \quad \mathsf{transcript}_{\mathsf{PRC}} = \varnothing.$$

  2. As many times as desired, the adversary submits the following:
     (a) On an encoding query, the challenger samples $z \xleftarrow{R} \{0,1\}^\lambda$. If $\mathsf{transcript}_{\mathsf{PRF}}$ contains no entry with first component z, it samples $(R_{1,z}, R_{2,z}) \xleftarrow{R} \{0,1\}^\lambda \times \{0,1\}^{r(\lambda)}$ and sets

     $$\mathsf{transcript}_{\mathsf{PRF}} := \mathsf{transcript}_{\mathsf{PRF}} \cup \{(z, (R_{1,z}, R_{2,z}))\}.$$

     Otherwise, it retrieves the stored pair $(R_{1,z}, R_{2,z})$. In either case, it computes

     $$c_z \leftarrow \Pi_{\mathsf{list}}.\mathsf{Encode}(\mathsf{sk}_{\mathsf{list}}, (z, R_{1,z}); R_{2,z}).$$

     It sets $\mathsf{transcript}_{\mathsf{PRC}} := \mathsf{transcript}_{\mathsf{PRC}} \cup \{(z, c_z)\}$. It returns c_z to the adversary.
     (b) On a decoding query c', the challenger returns 1 if there exists a pair (z, c_z) in $\mathsf{transcript}_{\mathsf{PRC}}$ satisfying $\delta(c_z, c') \leq p$, and $\perp$ otherwise.
  3. The adversary returns a bit $\hat{b}$, the output of the experiment.

- $\mathsf{Hyb}_{\mathsf{ideal}}$: In this experiment, we replace PRC codewords with random strings. This is the ideal security experiment.

  1. The challenger initializes $\mathsf{transcript}_{\mathsf{PRC}} = \varnothing$.
  2. As many times as desired, the adversary submits the following:
     (a) On an encoding query, the challenger samples $z \xleftarrow{R} \{0,1\}^\lambda$. If $\mathsf{transcript}_{\mathsf{PRC}}$ contains no entry with first component z, it samples $c_z \xleftarrow{R} \{0,1\}^{t(\lambda)}$ and sets $\mathsf{transcript}_{\mathsf{PRC}} := \mathsf{transcript}_{\mathsf{PRC}} \cup \{(z, c_z)\}$. Otherwise, it retrieves the stored value c_z. In either case, it returns c_z to the adversary.
     (b) On a decoding query c', the challenger proceeds as in Hyb_2.
  3. The adversary returns a bit $\hat{b}$, the output of the experiment.

Now we observe that for any efficient adversary $\mathcal{A}$ in the above hybrids, the output distributions of adjacent hybrids are negligibly close.

From $\mathsf{Hyb}_{\mathsf{real}}$ to Hyb_1. This follows from a straightforward reduction to PRF security. Since the adversary $\mathcal{A}$ and the list decoding $\Pi_{\mathsf{list}}.\mathsf{Decode}$ are efficient, they can only induce a polynomial number of PRF queries.

From Hyb_1 to Hyb_2. We can argue this query-by-query. Consider the ith decoding query, where we have replaced the first $i-1$ decoding queries with Hyb_2. It suffices to show the following:

1. Except with negligible probability, Hyb_1 returns 1 in this query whenever Hyb_2 does: for any $(z, c_z) \in \mathsf{transcript}_{\mathsf{PRC}}$ with $\delta(c_z, c') \leq p$, the associated message $(z, R_{1,z})$ is recovered by $\Pi_{\mathsf{list}}.\mathsf{Decode}$.

2. When Hyb_2 returns $\perp$ in this query, Hyb_1 returns 1 only with negligible probability.

Enabling both arguments is the observation that after the first $i-1$ decoding queries are replaced with Hyb_2, the execution of the experiment before query i is independent of $\Pi_{\mathsf{list}}.\mathsf{Decode}$. Thus, 1 follows from ordinary p-robustness of Π_{list}.

On the other hand, 2 follows from the fact that if an adversary is able to induce a different response on this decoding query, then there is some message (z^*, R_1^*) recovered by $\Pi_{\mathsf{list}}.\mathsf{Decode}$ in the decoding list T that passes the check $R_1^* = R_{1,z^*}$. However, since z^* was not sampled in any previous encoding query, R_{1,z^*} is a uniform random string independent of the experiment thus far, and thus equal to R_1^* with only $2^{-\lambda}$ probability. We can union bound over all polynomially many elements of T, and apply a hybrid argument over all polynomially many decoding queries made by the efficient adversary $\mathcal{A}$ to argue indistinguishability.

From Hyb_2 to $\mathsf{Hyb}_{\mathsf{ideal}}$. This follows from a straightforward reduction to pseudorandomness of the list-decodable PRC. In both experiments, decoding queries are answered using only $\mathsf{transcript}_{\mathsf{PRC}}$. Once PRC codewords are replaced by random strings, all of the adversary's queries are answered independently of the list-decoding secret key $\mathsf{sk}_{\mathsf{list}}$, which can now be erased.

□

Acknowledgments. The author was partially supported by an Amazon Research Award, Google CyberNYC, NSF CCF-2312224, NSF CCF-2107187, NSF CCF-2332922, the UK AISI Alignment Project, and The Leighton Family Chair.

This survey was written largely while the author was a student at Columbia University, and a partial version appeared in her Ph.D. thesis. This survey reflects countless conversations and insightful questions from the community. Many of these conversations were with Sam Gunn. The author would also like to thank Daniel Wichs, Aayush Jain, Noah Golowich, Tal Malkin, Yevgeniy Dodis, Alex Hoover, Alon Rosen, Adam O'Neill, Yuval Ishai, Mahesh Sreekumar Rajasree, and Surendra Ghentiyala.

The construction of low-rate edit robustness from one-way functions is from a conversation with Sam Gunn and Daniel Wichs.

The author thanks George Lu and Jad Silbak for the solution to Question 7 and for allowing the inclusion of its proof.

AI Usage. AI was used for proofreading and making figures.

References

[AAC+25] Omar Alrabiah, Prabhanjan Ananth, Miranda Christ, Yevgeniy Dodis, and Sam Gunn. Ideal pseudorandom codes. In Michal Koucký and Nikhil Bansal, editors, *Proceedings of the 57th Annual ACM Symposium on Theory of Computing, STOC 2025, Prague, Czechia, June 23-27, 2025*, pages 1638–1647. ACM, 2025.

[ASS+23] Shweta Agrawal, Sagnik Saha, Nikolaj Ignatieff Schwartzbach, Akhil Vanukuri, and Prashant Nalini Vasudevan. k-sum in the sparse regime. Cryptology ePrint Archive, Paper 2023/488, 2023. `https://eprint.iacr.org/2023/488`.

[BHMW21] Elette Boyle, Justin Holmgren, Fermi Ma, and Mor Weiss. On the security of doubly efficient PIR. *IACR Cryptol. ePrint Arch.*, 2021:1113, 2021.

[BIPW17] Elette Boyle, Yuval Ishai, Rafael Pass, and Mary Wootters. Can we access a database both locally and privately? In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, Lecture Notes in Computer Science, pages 662–693. Springer, 2017.

[BKR23] Andrej Bogdanov, Pravesh K. Kothari, and Alon Rosen. Public-key encryption, local pseudorandom generators, and the low-degree method. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part I*, Lecture Notes in Computer Science, pages 268–285. Springer, 2023.

[Blu03] Avrim Blum. Learning a function of r relevant variables. In Bernhard Schölkopf and Manfred K. Warmuth, editors, *Computational Learning Theory and Kernel Machines, 16th Annual Conference on Computational Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003, Proceedings*, volume 2777 of *Lecture Notes in Computer Science*, pages 731–733. Springer, 2003.

[BW21] Keller Blackwell and Mary Wootters. A note on the permuted puzzles toy conjecture. *CoRR*, abs/2108.07885, 2021.

[CCP17] Alain Couvreur, Irene Marquez Corbella, and Ruud Pellikaan. Cryptanalysis of mceliece cryptosystem based on algebraic geometry codes and their subcodes. *IEEE Trans. Inf. Theory*, 63(8):5404–5418, 2017.

[CG24] Miranda Christ and Sam Gunn. Pseudorandom error-correcting codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VI*, volume 14925 of *Lecture Notes in Computer Science*, pages 325–347. Springer, 2024.

[CGG+14] Alain Couvreur, Philippe Gaborit, Valérie Gauthier-Umaña, Ayoub Otmani, and Jean-Pierre Tillich. Distinguisher-based attacks on public-key cryptosystems using reed-solomon codes. *Des. Codes Cryptogr.*, 73(2):641–666, 2014.

[CGG+25] Miranda Christ, Noah Golowich, Sam Gunn, Ankur Moitra, and Daniel Wichs. Improved pseudorandom codes from permuted puzzles. *arXiv preprint arXiv:2512.08918*, 2025.

[CGZ24] Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In Shipra Agrawal and Aaron Roth, editors, *The Thirty Seventh Annual Conference on Learning Theory, June 30 - July 3, 2023, Edmonton, Canada*, Proceedings of Machine Learning Research, pages 1125–1139. PMLR, 2024.

[CHR17] Ran Canetti, Justin Holmgren, and Silas Richelson. Towards doubly efficient private information retrieval. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography - 15th International Conference, TCC 2017, Baltimore, MD, USA, November 12-15, 2017, Proceedings, Part II*, volume 10678 of *Lecture Notes in Computer Science*, pages 694–726. Springer, 2017.

[CMY26] Yu Chen, Xinyu Mao, and Hongxu Yi. Advancing pseudorandom codes: Beyond parity checks and standard-model CCA1 security. *IACR Cryptol. ePrint Arch.*, 2026:1228, 2026.

[DFV26] Victor Dyseryn, Danilo Francati, and Daniele Venturi. Public-key pseudorandom codes from distorted mceliece assumptions. *IACR Cryptol. ePrint Arch.*, 2026:1184, 2026.

[DJK+26] Nico Döttling, Antoine Joux, Venkata Koppula, Mahesh Sreekumar Rajasree, and Hendrik Waldner. Chosen ciphertext secure pseudorandom codes in the standard model. *IACR Cryptol. ePrint Arch.*, 2026:1227, 2026.

[DMR25] Nico Döttling, Anne Müller, and Mahesh Sreekumar Rajasree. Separating pseudorandom codes from local oracles. In Benny Applebaum and Huijia (Rachel) Lin, editors, *Theory of Cryptography - 23rd International Conference, TCC 2025, Aarhus, Denmark, December 1-5, 2025, Proceedings, Part IV*, Lecture Notes in Computer Science, pages 225–257. Springer, 2025.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Annual international cryptology conference*, pages 537–554. Springer, 1999.

[GG25] Surendra Ghentiyala and Venkatesan Guruswami. New constructions of pseudorandom codes. In Alina Ene and Eshan Chattopadhyay, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2025, Berkeley, CA, USA, August 11-13, 2025*, LIPIcs, pages 54:1–54:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

[GGW25] Sanjam Garg, Sam Gunn, and Mingyuan Wang. Black-box crypto is useless for pseudorandom codes. In Benny Applebaum and Huijia (Rachel) Lin, editors, *Theory of Cryptography - 23rd International Conference, TCC 2025, Aarhus, Denmark, December 1-5, 2025, Proceedings, Part IV*, Lecture Notes in Computer Science, pages 205–224. Springer, 2025.

[GM24] Noah Golowich and Ankur Moitra. Edit distance robust watermarks via indexing pseudorandom codes. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang, editors, *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024.

[Gol11] Oded Goldreich. Candidate one-way functions based on expander graphs. *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation: In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, pages 76–87, 2011.

[GZS25] Sam Gunn, Xuandong Zhao, and Dawn Song. An undetectable watermark for generative image models. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[HLLL25] Xuming Hu, Hanqian Li, Jungang Li, and Aiwei Liu. Videomark: A distortion-free robust watermarking framework for video diffusion models. *CoRR*, abs/2504.16359, 2025.

[HLMZ26] Shengtang Huang, Xin Li, Songtao Mao, and Zhaienhe Zhou. High-rate public-key pseudorandom codes for edit errors. *arXiv preprint arXiv:2605.19402*, 2026.

[HLVA02] Nicholas J Hopper, John Langford, and Luis Von Ahn. Provably secure steganography. In *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*, pages 77–92. Springer, 2002.

[HvL98] Tom Høholdt and van Lint. Algebraic geometry codes. In *Handbook of Coding Theory*, pages 871–962, United Kingdom, 1998. Elsevier.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007.

[KTHL24] Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. Robust distortion-free watermarks for language models. *Trans. Mach. Learn. Res.*, 2024, 2024.

[LMW25] Wei-Kai Lin, Ethan Mook, and Daniel Wichs. Black box crypto is useless for doubly efficient PIR. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part VI*, volume 15606 of *Lecture Notes in Computer Science*, pages 65–93. Springer, 2025.

[LSS26] Huijia Lin, Kameron Shahabi, and Min Jae Song. Unforgeable watermarks for language models via robust signatures. *arXiv preprint arXiv:2602.15323*, 2026.

[LSW26] George Lu, Jad Silbak, and Daniel Wichs. Achieving shannon capacity for computationally bounded errors. *IACR Cryptol. ePrint Arch.*, 2026:1221, 2026.

[MMP26] Noam Mazor, Andrew Morgan, and Rafael Pass. Can we watermark low-entropy LLM outputs? In Huijia (Rachel) Lin, editor, *7th Symposium on Foundations of Responsible Computing, FORC 2026, Harvard University, Cambridge, MA, USA, June 3-5, 2026*, volume 368 of *LIPIcs*, pages 8:1–8:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026.

[RBL+22] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 10674–10685. IEEE, 2022.

[SKS19] Jad Silbak, Swastik Kopparty, and Ronen Shaltiel. Quasilinear time list-decodable codes for space bounded channels. In David Zuckerman, editor, *60th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2019, Baltimore, Maryland, USA, November 9-12, 2019*, pages 302–333. IEEE Computer Society, 2019.

[WWC+25] Tianrui Wang, Anyu Wang, Tianshuo Cong, Delong Ran, Jinyuan Liu, and Xiaoyun Wang. Cryptanalysis of pseudorandom error-correcting codes. *CoRR*, abs/2512.17310, 2025.

[YZC+24] Zijin Yang, Kai Zeng, Kejiang Chen, Han Fang, Weiming Zhang, and Nenghai Yu. Gaussian shading: Provable performance-lossless image watermarking for diffusion models. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pages 12162–12171. IEEE, 2024.

[YZC+25] Zijin Yang, Xin Zhang, Kejiang Chen, Kai Zeng, Qiyi Yao, Han Fang, Weiming Zhang, and Nenghai Yu. Gaussian shading++: Rethinking the realistic deployment challenge of performance-lossless image watermark for diffusion models. *CoRR*, abs/2504.15026, 2025.

[ZZJ+26] Xinjie Zhu, Zijing Zhao, Hui Jin, Qingxiao Guo, Yilong Ma, Yunhao Wang, Xiaobing Guo, and Weifeng Zhang. Sigmark: Scalable in-generation watermark with blind extraction for video diffusion. *CoRR*, abs/2603.02882, 2026.