Pseudorandom Error-Correcting Codes

Miranda Christ[1(✉)] and Sam Gunn[2]

[1] Columbia University, New York, USA
mchrist@cs.columbia.edu
[2] UC Berkeley, Berkeley, USA
gunn@berkeley.edu

Abstract. We construct *pseudorandom error-correcting codes* (or simply *pseudorandom codes*), which are error-correcting codes with the property that any polynomial number of codewords are pseudorandom to any computationally-bounded adversary. Efficient decoding of corrupted codewords is possible with the help of a decoding key.

We build pseudorandom codes that are robust to substitution and deletion errors, where pseudorandomness rests on standard cryptographic assumptions. Specifically, pseudorandomness is based on either $2^{O(\sqrt{n})}$-hardness of LPN, or polynomial hardness of LPN and the planted XOR problem at low density.

As our primary application of pseudorandom codes, we present an undetectable watermarking scheme for outputs of language models that is robust to cropping and a constant rate of random substitutions and deletions. The watermark is undetectable in the sense that any number of samples of watermarked text are computationally indistinguishable from text output by the original model. This is the first undetectable watermarking scheme that can tolerate a constant rate of errors.

Our second application is to steganography, where a secret message is hidden in innocent-looking content. We present a constant-rate stateless steganography scheme with robustness to a constant rate of substitutions. Ours is the first stateless steganography scheme with provable steganographic security and *any* robustness to errors.

1 Introduction

The proliferation of AI-generated content is one of the biggest issues facing the internet today. Recent breakthroughs in large language models have made it increasingly difficult to distinguish this influx of AI-generated content from human-generated content.

A promising solution for detecting AI-generated content is *watermarking*, where a hidden signal is embedded in the content. Several major companies, including OpenAI and Google, have pledged to embed watermarks in content output by their models [7]. Despite this explosion of interest in watermarking, there are very few techniques for building watermarking schemes that do not noticeably alter the generated content. Existing schemes incur trade-offs

L. Reyzin and D. Stebila (Eds.): CRYPTO 2024, LNCS 14925, pp. 325–347, 2024.
https://doi.org/10.1007/978-3-031-68391-6_10

between the quality of generated content, the robustness of the watermark, and the computational complexity of detection.

In this work, we take a new cryptographic approach to this problem that allows us to avoid some of these trade-offs. Our approach is based on a new cryptographic primitive that we call a pseudorandom error-correcting code, or simply a pseudorandom code (PRC). A PRC is an error-correcting code that is parameterized by a decoding key. The pseudorandomness property states that, without this decoding key, any polynomial number of codewords are pseudorandom.

We find that the problem of building robust, quality-preserving watermarks reduces to the problem of building PRCs. Essentially, the watermarking strategy is to replace some of the randomness used by the generative algorithm with outputs from a PRC.

Building PRCs is challenging: Error-correcting codes are typically highly structured, while pseudorandomness implies a lack of discernible structure. Indeed, a priori it is not clear that such objects should exist. Nonetheless, we construct PRCs from standard (subexponential) cryptographic assumptions. Our constructions are related to low-density parity-check codes, and we base pseudorandomness on the Learning Parity with Noise assumption. We construct PRCs with strong robustness properties, including robustness to any constant rate of substitution and deletion errors.

Applying these PRCs to watermarking for language models, we obtain the first quality-preserving language model watermarking schemes that are robust to cropping and any constant rate of substitution and deletion errors. That is, the watermark detector will work as long as it is provided any sufficiently-long sequence of text, even if that text is subjected to any constant (less than 1/2) rate of random substitutions and any constant (less than 1) rate of random deletions.

1.1 An Approach to Watermarking

In this subsection we present a simple, general template for watermarking AI-generated content. The template described here can in principle be used to watermark arbitrary media, but we only present concrete instantiations in certain contexts.

In all generative AI settings there is a generative algorithm, $\mathsf{Generate}$, that defines the behavior of the AI. A user provides a prompt, and $\mathsf{Generate}$ outputs some randomly-generated content. A watermarking scheme modifies $\mathsf{Generate}$ so that the generated content contains a hidden pattern, called a watermark. There are two essential requirements that any watermarking scheme should satisfy:

- *Quality*: embedding the watermark should not reduce the quality of generative algorithm; and
- *Robustness*: the watermark should be detectable in generated content, even if this content is corrupted.

Achieving both of these properties simultaneously is the central challenge of watermarking. Quality means that the watermark should not significantly alter the generated content, while robustness seems to require the watermark to change the content a great deal.

In this work, we propose a new strategy for watermarking: *replacing the randomness used by* Generate *with codewords from a pseudorandom error-correcting code*. A pseudorandom error-correcting code, or simply pseudorandom code (PRC), is a new cryptographic primitive that we introduce in this work. A PRC is defined by algorithms $\mathsf{Encode}, \mathsf{Decode}$ satisfying two properties:

- *Pseudorandomness*: Any efficient adversary, without knowledge of the decoding key, cannot distinguish between oracle access to Encode and an oracle that always outputs a fresh random string; and
- *Error correction* or *robustness*: For any message m, if $x \leftarrow \mathsf{Encode}(\mathsf{m})$ and x' is a "corrupted" version of x where the amount of corruption is bounded, then $\mathsf{Decode}(x') = \mathsf{m}$.

For watermarking the message can be simply $\mathsf{m} = 1$, indicating the presence of a watermark.[1]

In order to make detection possible, we specify Generate in such a way that the detector can approximate the randomness used to produce any given content. To test for the presence of a watermark, the detector computes this approximation and then applies Decode to the result. If the content is watermarked and the approximation is close enough to the true randomness, then robustness of the PRC ensures that Decode returns 1. This indicates to the detector that the watermark is present. Stronger robustness of the PRC translates to stronger robustness of the watermark.

Pseudorandomness of the PRC guarantees that, without the decoding key, watermarked content is indistinguishable from content produced by the original generative algorithm—even if one is allowed to see many outputs. In particular, the quality of the generative algorithm is not deteriorated by the watermark. In [8], such a watermark is referred to as *undetectable*.

Therefore, the problem of building robust, quality-preserving watermarks reduces to building PRCs (and appropriately specifying Generate). Below, we describe the application of this template to watermarking for language models.

Watermarks for Language Models. A generative language model is a randomized algorithm that takes as input a prompt, and samples text constituting a response. This text consists of *tokens*. For simplicity, we assume here that tokens are binary and that the full response is of a fixed length n. Neither of these assumptions is important for our results, as discussed in the full version.

Given any generative language model, it is not difficult to define an algorithm Generate that takes a prompt and a random seed $x \in \{0,1\}^n$, and samples a response $\mathsf{t} \in \{0,1\}^n$ such that

[1] By instead encoding a longer message in the PRC, this technique extends to steganography—where messages are secretly communicated in innocent-looking content—as well.

– if x is uniformly random, then t is distributed identically to the original model, and
– each bit of t is correlated with the corresponding bit of x.

See Sect. 2.5 for an example of such an algorithm. Now it is easy to recover an approximation of the randomness x that was used to produce a given text: just use the text t itself.

One natural watermarking strategy is to use the *same* random seed $x \in \{0,1\}^n$ for every call to $\mathsf{Generate}$, storing x itself as the watermarking key. This is essentially the strategy used by [13], and it yields a highly robust watermark because the detector can compute the edit distance between the given text and x. The resulting quality guarantee is that a single response from the watermarked model is distributed identically to a single response from the original model.

However, this strategy results in redundancy of responses because the i^{th} token of every response is biased towards the i^{th} bit of x. This is problematic as it limits the variability of text that the language model generates. For instance, it should not be the case that certain words are preferred as the first word of every response. One can mitigate this issue by storing a family of seeds $x_1, \ldots, x_\ell \in \{0,1\}^n$ and randomly choosing one such seed for each response. Increasing ℓ improves the variability, but comes at the cost of a corresponding increase in both watermark key length *and* detector runtime. Now, the detector must compute the edit distance for each seed, resulting in a runtime of $O(n^2\ell)$. In particular, for any polynomial-time watermarking scheme using this approach, there exists a polynomial-time adversary that can distinguish watermarked content from unwatermarked content without needing the watermark detection key.

A PRC is exactly the object needed to avoid this tradeoff between response variability and efficiency. Our watermark detection key will be the decoding key for a PRC, and we will embed the watermark by sampling a fresh pseudorandom seed $x \leftarrow \mathsf{Encode}(1)$ for each query to $\mathsf{Generate}$. This results in no observable correlations between responses, regardless of the number of queries—i.e., the watermark is undetectable. Since the same hidden structure is present in every sample from $\mathsf{Encode}(1)$, our detector can simply apply Decode to check for this structure. Now, the detector's runtime has no dependence on the response variability. Using PRCs that we construct, we also find that our watermarking scheme can be made robust to a constant fraction of random substitution and deletion errors.

A Note on Undetectability. Undetectability is a strong quality guarantee for watermarking. Outputs of the watermarked model must be *computationally indistinguishable* from outputs of the original model, even to a user who is allowed to make many queries. While this is imperative for steganography, its necessity in watermarking is less clear, as some noticeable changes to the model may be permissible as long as the outputs remain "high-quality."

However, measuring the quality of watermarked content is often challenging or impossible—especially when the content is used in a wide range of applications. Computational indistinguishability is a strong quality guarantee that

applies uniformly to every application: it implies that the watermark causes no observable loss in *any* efficiently-computable quality metric. Without such a guarantee, it is impossible to verify that a watermark is quality-preserving across all applications. We therefore focus on undetectability in this work.

1.2 Our Contributions

Pseudorandom Codes. Our first contribution is to identify PRCs as an interesting cryptographic primitive, with applications to robustly hiding messages in AI-generated content. Very roughly, the definition is as follows—for more details, see the technical overview (Sect. 2.1).

Definition. *A* pseudorandom code *(PRC) is an error-correcting code where codewords are pseudorandom to any computationally-bounded adversary who doesn't hold the decoding key.*

We consider both public- and secret-key variants of PRCs. For a public-key PRC, the encoding algorithm can be made public without sacrificing pseudorandomness. When the message space consists of only a single message, we call it a *zero-bit* PRC. Zero-bit PRCs can also be viewed as robust *backdoored (or trapdoor) pseudorandom generators* [9,17], and they are sufficient for our applications to watermarking.

We show how to build zero-bit public-key PRCs related to low-density parity-check (LDPC) codes, where pseudorandomness rests on standard (subexponential) cryptographic assumptions. All of the PRCs we construct are over a binary alphabet. Depending on the parameter choices, the pseudorandomness of these LDPC-related codes is based on either of two assumptions, which we state together as Assumption 1.

Assumption 1. *Either*

- *LPN is hard for any $2^{O(\sqrt{n})}$-time adversary, or*
- *LPN and planted XOR are both hard for any polynomial-time adversary.*

For descriptions of the LPN and planted XOR assumptions, see Sect. 2.2. Under Assumption 1, we prove that there exist PRCs with robustness to channels that introduce a bounded number of substitution errors. We say that any channel that introduces at most a p fraction of substitution errors (and no other types of errors) is *p-bounded*.

Theorem 1. *Let $p \in (0, 1/2)$ be any constant. Under Assumption 1, there exists a zero-bit public-key PRC that is robust to every p-bounded channel.*

For some applications it will be useful to have multi-bit PRCs. Any such construction should ideally have a high *rate*, which is the ratio of the number of message bits to the number of codeword bits. We prove that any zero-bit PRC can be combined with any error correcting code to give a multi-bit PRC.

Theorem 2. *Suppose that there exists a zero-bit (public-key) PRC and a rate-R error-correcting code, that are both robust to every p-bounded channel. Then there exists a (public-key) PRC of rate $R - o(1)$ that is robust to every $(p-\varepsilon)$-bounded channel, for every constant $\varepsilon > 0$.*

Applying this theorem with our zero-bit LDPC-based PRCs and the binary error-correcting codes of [5,14,16], we have the following corollary.

Corollary. *Let $p \in (0, 1/2)$ be any constant. Under Assumption 1, there exists a constant-rate PRC that is robust to every p-bounded channel.*

None of the PRCs mentioned so far can handle deletions. Deletions are a particularly important type of edit for text watermarks, because an adversary may try to remove the watermark by simply deleting some of the words.

Deletions are notoriously difficult to handle in error correction, and standard techniques involve significant structure—thus violating pseudorandomness. Nonetheless, we show that if a PRC has sufficiently strong robustness to substitutions, then it can be converted to a PRC with robustness to deletions (at the cost of a decreased rate).

Let BSC_p be the binary symmetric channel with error rate p, and BDC_q be the binary deletion channel with deletion rate q. That is, BSC_p randomly flips each bit with probability p and BDC_q randomly deletes each bit with probability q.

Theorem 3. *For any constants $p \in (0, 1/2)$ and $q \in (0, 1)$, there exists $p' \in (0, 1/2)$ such that the following holds. If there exists a zero-bit (public-key) PRC with robustness to every p'-bounded channel, then there exists a zero-bit (public-key) PRC that is robust to the channel $BSC_p \circ BDC_q$.*

Together with our LDPC-based PRCs, we obtain the following result.

Corollary. *Let $p \in (0, 1/2)$ and $q \in (0, 1)$ be any constants. Under Assumption 1, there exists a zero-bit public-key PRC that is robust to the channel $BSC_p \circ BDC_q$.*

Watermarking and Steganography for Language Models. We apply our zero-bit PRCs to build the first undetectable watermarking scheme for language models with robustness to a constant rate of random substitutions and deletions. In this section, we assume that text output by the language model is represented as a bitstring. Arbitrary text can be mapped to a bitstring by either randomly assigning a single bit to each token, or by expanding the tokens into a binary representation.

Theorem 4. *Let $p \in (0, 1/2)$ be any constant. Under Assumption 1, there exists an undetectable watermarking scheme $\mathcal{W}$ such that the watermark appears in any sufficiently high-entropy text, even if the text is subjected to the channel BSC_p.*

Under an extra assumption about the generated text, which roughly corresponds to the text having few repeated words, we can strengthen this to handle deletions as well.

Theorem 5. *Let $p \in (0, 1/2)$ and $q \in (0, 1)$ be any constants. Under Assumption 1, there exists an undetectable watermarking scheme $\mathcal{W}$ such that the watermark appears in any sufficiently high-entropy and "variable" text, even if the text is subjected to the channel $BSC_p \circ BDC_q$.*

In all of our theorems the text can be cropped, as long as the remaining text is sufficiently high-entropy.

We also construct undetectable watermarking schemes with *unforgeable public attribution* and the same robustness as $\mathcal{W}$. Public attribution means that there is a public algorithm to identify which portion of a given text was output by the model. Unforgeability means that no efficient user can produce text that the attribution algorithm identifies as model-generated, but that was not output by the model. Interestingly, our schemes retain the standard robust secret-key detector in addition to this public attribution algorithm.

Theorem 6. *Under Assumption 1, there exists a watermarking scheme $\mathcal{W}_{\mathsf{att}}$ that retains all properties of $\mathcal{W}$ from Theorem 4 or Theorem 5, and additionally satisfies unforgeable public attribution.*

Finally, using PRCs with constant rate (Theorem 2), we also obtain the first *robust* language model steganography scheme.[2]

Theorem 7. *Let $p \in (0, 1/2)$ be any constant. Under Assumption 1, there exists a language model steganography scheme with constant information rate, such that the message can be recovered from any sufficiently high-entropy text, even if the text is subjected to the channel BSC_p.*

Universal Steganography. We show that PRCs can be used to solve a long-standing open question in steganography: A simple application of PRCs yields the first robust, stateless universal steganography scheme. Universal steganography can be used for language model steganography, but it is more general [3]. We take the rate of the steganography scheme to be the ratio of the number of stegotext symbols to the number of bits in the message being encoded.

Theorem 8. *Suppose there is a hash function that is unbiased over the covertext channel. If PRC is any PRC, then there exists a stateless, public-key universal steganography scheme with the same rate and robustness as PRC.*

Finally, we show that this result can be extended to the setting where an unbiased hash function on the covertext channel is not known, with some loss in robustness.

[2] Since this scheme doesn't rely on the decoder having access to the prompt, it can also be seen as an undetectable "multi-bit watermarking scheme".

Theorem 9. *Suppose there is a hash function that has constant min-entropy over the covertext channel. Then under Assumption 1, for any $p \in (0, 1/2)$, there exists a constant-rate public-key stateless steganography scheme that is robust to a p rate of random substitutions.*

1.3 Related Work: Short Summary

We briefly outline some of the related work here. See the full version for a more complete discussion.

- Code-based cryptography: Our work bears some similarity to the field of code-based cryptography. However, code-based cryptography is generally focused on building existing primitives from new assumptions—whereas PRCs are a new primitive that we base on existing assumptions.
- Trapdoor pseudorandom generators: Our zero-bit PRCs can be equivalently viewed as *robust* trapdoor (or equivalently, backdoored) pseudorandom generators [9,17]. That is, we require the additional property that the trapdoor (or secret key) can be used to detect even *corrupted* pseudorandom strings.
- Watermarking for language models: We build watermarking schemes for language models satisfying the strongest quality guarantee, undetectability. Undetectability was defined by [8], where undetectable watermarks for language models were also constructed. In that work and in [1,12], it is essential for watermark detection that many sufficiently-long contiguous substrings of the response remain *unchanged*. Therefore, these watermarks are easily removable by simple attacks (see the "robustness of our watermark" paragraph of Sect. 2.5). The watermarks of [13] are more robust—their robustness is more comparable to ours—but they sacrifice undetectability. Instead, their watermarks satisfy the weaker property of distortion-freeness, which is the single-query version of undetectability. [19] obtain even stronger robustness, at the cost of even further reduced quality.
- Impossibility of strong watermarks: [18] explore the possibility of watermarking for language models in the presence of motivated adversaries. They argue that sampling a *random* response is easier when one is provided *any* response. Since a random response cannot be watermarked (or else there would be a high false-positive rate), they use this to argue that any watermarked language model necessarily provides some assistance in generating un-watermarked text.
- Steganography: Steganography is the study of concealing secret messages in innocent-looking content. Whereas encryption is about hiding the message, steganography is about hiding the *existence* of the message. Ever since steganography was formalized by [11], *robust* steganography schemes (that don't require a shared state) have remained elusive. We resolve this problem using PRCs.

2 Technical Overview

2.1 Pseudorandom Code Basics

Pseudorandom codes (PRCs) can be viewed as a combination of two related primitives:

- Pseudorandom encryption, where ciphertexts are indistinguishable from random under a chosen plaintext attack [15]. Secret-key pseudorandom encryption is easy to build using a pseudorandom function F_1—just encrypt m by sampling a random r and outputting $(r, \mathsf{m} \oplus F_1(r))$. Public-key pseudorandom encryption is also known from standard assumptions [3]. However, none of these constructions have any nontrivial robustness.
- Robust encryption, where encryptions of messages are robust to errors. Robust encryption is easy to build by applying an error-correcting code to ciphertexts from any standard encryption scheme. Even if that encryption scheme is pseudorandom, the use of the error-correcting code will in general render the robust encryption scheme not pseudorandom.

A PRC is required to simultaneously satisfy *both* pseudorandomness and robustness—properties that are in direct tension with each other. Using the secret key, one should be able to discern the redundancy and structure that give ciphertexts their robustness. Without the secret key, ciphertexts must appear completely unstructured.

We define secret-key PRCs below. For public-key PRCs, we further require that the encoding algorithm can be made public without sacrificing pseudorandomness.

Definition. *Let Σ be an alphabet and $\mathcal{E} : \Sigma^* \to \Sigma^*$ be a channel. A* secret-key PRC *with robustness to $\mathcal{E}$ is described by algorithms $\mathsf{Encode}_1 : \Sigma^k \to \Sigma^n$ and $\mathsf{Decode}_1 : \Sigma^* \to \Sigma^k \cup \{\perp\}$, parameterized by a secret key $\mathsf{1}$, satisfying the following criteria for every security parameter λ:*

- *(Error correction, or robustness) For any message $\mathsf{m} \in \Sigma^k$,*

$$\Pr_{\mathsf{1}}[\mathsf{Decode}_1(\mathcal{E}(x)) = \mathsf{m} : x \leftarrow \mathsf{Encode}_1(\mathsf{m})] \geq 1 - \mathsf{negl}(\lambda).$$

- *(Soundness) For any fixed $c \in \Sigma^*$,*

$$\Pr_{\mathsf{1}}[\mathsf{Decode}_1(c) = \perp] \geq 1 - \mathsf{negl}(\lambda).$$

- *(Pseudorandomness) For any polynomial-time adversary $\mathcal{A}$,*

$$|\Pr_{\mathsf{1}}[\mathcal{A}^{\mathsf{Encode}_1}(1^\lambda) = 1] - \Pr_{\mathcal{U}}[\mathcal{A}^{\mathcal{U}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda),$$

where $\mathcal{A}^{\mathcal{U}}$ means that the adversary has access to an oracle that, on any (even previously queried) input, responds with a freshly drawn uniform value in Σ^n.

If the scheme can only encode a singular message (i.e. $k = 0$), then we call it a *zero-bit* PRC. Soundness is a technical condition that we include only to ensure that zero-bit PRCs are non-trivial.

For a sufficiently weak channel $\mathcal{E}$, it is not hard to construct a secret-key PRC with robustness to $\mathcal{E}$ where pseudorandomness rests on very mild assumptions. For instance, if $F_1 : \{0,1\}^\ell \to \{0,1\}$ is a pseudorandom function, we can build a zero-bit secret-key PRC with the following encoding algorithm:

$\mathsf{Encode}_1(1)$:

1. Randomly sample $x_1, \ldots, x_\ell \leftarrow \{0,1\}$.
2. For $i = \ell + 1, \ldots, n$, let $x_i = F_1(x_{i-\ell}, \ldots, x_{i-1})$.
3. Output $x_1, \ldots, x_n$.

The decoding algorithm Decode_1 simply checks whether much more than a $1/2$ fraction of conditions $x_i = F_1(x_{i-\ell}, \ldots, x_{i-1})$ are satisfied. Pseudorandomness follows by taking ℓ to be the security parameter. It is immediate that this PRC is robust to any length-preserving channel that introduces at most a p fraction of errors, for $p << 1/\ell$. We call any such channel *p-bounded*.

This secret-key PRC construction can be seen as implicit in prior hash-based watermarking schemes [1,8,12], where essentially the same level of robustness to a $1/\ell$ error rate is obtained. Unfortunately, it has an inherent trade-off between pseudorandomness and robustness: After roughly $2^{\ell/2}$ samples from $\mathsf{Encode}_1(1)$, there will be repeated prefixes and therefore correlations between the samples. In particular, if we want this PRC to be robust to a constant rate of errors, we have to set $\ell = O(1)$, in which case even a constant number of queries are enough to observe correlations. We therefore turn to alternative constructions.

2.2 Pseudorandom LDPC Codes

Fortunately, one of the most prominent assumptions in theoretical cryptography is a statement about codes: Learning Parity with Noise (LPN). The LPN assumption states that noisy samples from the codespace of a random linear code are pseudorandom, even to an adversary who knows a generator matrix for the code. In more detail, let n, g be integers and $G \leftarrow \mathbb{F}_2^{n \times g}$ be a random matrix. The LPN assumption (with noise rate η and secrets of size g) states that $(G, Gs \oplus e) \approx (G, u)$,[3] where $s \leftarrow \mathbb{F}_2^g$, $e \leftarrow \mathrm{Ber}(n, \eta)$, and $u \leftarrow \mathbb{F}_2^n$.

The LPN assumption suggests using noisy codewords from a random linear code as a PRC. That is, let $G \leftarrow \mathbb{F}_2^{n \times g}$ be the secret key and Encode_G be the following zero-bit secret-key PRC encoder:

– $\mathsf{Encode}_G(1)$: Sample $s \leftarrow \mathbb{F}_2^g$ and $e \leftarrow \mathrm{Ber}(n, \eta)$. Output $Gs \oplus e$.

The LPN assumption immediately implies that an arbitrary polynomial number of samples from $\mathsf{Encode}_G(1)$ are pseudorandom. However, recall that the

[3] Throughout this work we use $\approx$ to refer to computational indistinguishability.

LPN assumption states that these samples are pseudorandom *even given* G—which means precisely that there does not exist an efficient zero-bit decoder $\mathsf{Decode}_G(x)$!

While this random linear code construction does not work, it naturally suggests a strategy that does. If we find a sampling procedure that produces a random (or even pseudorandom) generator matrix G *together with a trapdoor for efficient decoding*, then we have a public-key PRC where the generator matrix is the public encoding key and the trapdoor is the secret decoding key. By the LPN assumption, $\mathsf{Encode}_G(1)$ produces pseudorandom vectors even to an adversary who knows G, so the construction will satisfy pseudorandomness.

It turns out that low-weight parity checks can serve as such a trapdoor. That is, instead of sampling G uniformly at random, we first sample a "parity-check matrix" $P \in \mathbb{F}_2^{r \times n}$ with sparse rows (i.e., "low density"), and then sample $G \in \mathbb{F}_2^{n \times g}$ subject to $PG = 0$. For appropriate choice of n, g, t, r, we will show that the resulting marginal distribution on G is random or pseudorandom. The low-density parity-check matrix P will allow for efficient detection of near-codewords.

Codes defined by Low-Density Parity-Check matrices are called LDPC codes. For $n, g, t, r \in \mathbb{N}$, we define an (n, g, t, r) random LDPC code by the following distribution over parity-check and generator matrices:

$\mathsf{LDPC}[n, g, t, r]$:

1. Sample a random matrix $P \in \mathbb{F}_2^{r \times n}$ subject to every row of P being t-sparse.
2. Sample a random matrix $G \in \mathbb{F}_2^{n \times g}$ subject to $PG = 0$.
3. Output (P, G).

Zero-bit decoding works by counting the number of satisfied parity checks. For any fixed $x \in \mathbb{F}_2^n$, with high probability over $P \in \mathbb{F}_2^{r \times n}$ we expect that the number of unsatisfied parity checks, $\mathrm{wt}(Px)$, is roughly $r/2$. But if x is close to $\mathrm{Im}\, G \subseteq \ker P$ in Hamming distance, then as long as the error and the sparsity t of the parity checks are not too high, we expect $\mathrm{wt}(Px)$ to be significantly smaller than $r/2$.

Therefore our zero-bit pseudorandom LDPC code uses the following zero-bit decoding algorithm:

– $\mathsf{Decode}_P(x)$: If $\mathrm{wt}(Px) < (1/2 - r^{-1/4}) \cdot r$, output 1; otherwise output $\perp$.[4]

Encoding is exactly the same Encode_G algorithm as above—but now that G is sampled together with the trapdoor P, we have an efficient decoding algorithm. Observe that this is a zero-bit scheme, because the decoder only determines whether the input is related to the keys or not. Using belief propagation, it is possible to push this construction beyond a zero-bit PRC, although this results in lower robustness. We ultimately construct a constant-rate multi-bit PRC by other means, which we discuss in Sect. 2.4.

Let $\mathsf{LDPC\text{-}PRC}_0$ be the zero-bit public-key PRC defined by $(\mathsf{Encode}_G, \mathsf{Decode}_P)$ for $(P, G) \leftarrow \mathsf{LDPC}[n, g, t, r]$. In a moment we will outline our proofs

[4] Throughout this work, wt will refer to Hamming weight.

that $\mathsf{LDPC\text{-}PRC}_0$ is a public-key PRC with very strong robustness. First, let us see some restrictions on the sparsity parameter t that provide important context for these proofs.

If random noise of rate $1/2 - \varepsilon$ is applied to $x \in \operatorname{Im} G$, then the probability of each parity check being satisfied for the noisy codeword is $1/2 - (2\varepsilon)^t/2$. So in order for $\mathsf{Decode}_P(x)$ to output 1 with high probability, we need $(2\varepsilon)^t/2 > r^{-1/4}$, i.e., $t < 1 + \log r / 4\log(1/2\varepsilon) = O(\log r)$. We will always have $r = n^{\Omega(1)}$, so this restriction says that $t = O(\log n)$ for appropriate choice of constant.

On the other hand, if we set $t = O(1)$ then $\mathsf{Encode}_G(1)$ cannot be pseudorandom. This is because it is possible to brute-force search over all $\binom{n}{t}$ possible parity checks of weight t, and one can test whether Encode_G is consistent with a given parity check $s \in \mathbb{F}_2^n$ by simply computing $s \cdot x$ for many samples $x \leftarrow \mathsf{Encode}_G(1)$.

Therefore, we will choose $t = \Theta(\log n)$ in order to rely on the weakest possible cryptographic assumption for pseudorandomness, without sacrificing robustness to a constant noise rate.

Remark. The LDPC codes considered in this work differ from the traditional Gallager's LDPC ensemble in two important ways. First, our LDPC codes will have $t = \Theta(\log n)$ sparsity as opposed to constant sparsity. Unfortunately, the usual belief propagation decoder does not work for noise rates beyond $O(\log t/t)$; this is the reason why we only perform the simple zero-bit decoding. The second difference is that we use independent parity checks, which results in an irregular Tanner graph.

Remark. There is a well-known public-key encryption scheme, due to Alekhnovich [4, Cryptosystem 1], based on a low-noise variant of LPN. This scheme is similar to ours, but the decoder cannot tolerate any constant rate of errors.

Pseudorandom Generator Matrix. For appropriate choices of parameters, it turns out that the generator matrix of $\mathsf{LDPC}[n, g, t, r]$ is pseudorandom under the planted t-XOR assumption. The planted t-XOR problem (and its generalization, the planted t-SUM problem) is a natural and well-studied problem—see e.g. [2] for a more detailed discussion. Formally, the (n, m, t) *planted XOR problem* states that it is computationally hard to distinguish between

$\mathcal{D}_0^m$: a random m-dimensional linear subspace $V \subseteq \mathbb{F}_2^n$, and
$\mathcal{D}_1^m$: a random m-dimensional linear subspace $V_s \subseteq \mathbb{F}_2^n$ satisfying a random planted t-XOR relation s (i.e., s is a random t-sparse vector and $s \cdot v = 0$ for all $v \in V_s$).

Throughout this overview, we consider linear subspaces to be described by a random basis. Recalling the definition of $(P, G) \leftarrow \mathsf{LDPC}[n, g, t, r]$, if $r = 1$ the (n, g, t) planted XOR assumption immediately implies that G is pseudorandom (by identifying V with $\operatorname{Im} G$). But for the more interesting case that $r > 1$, we require a stronger version of the planted XOR assumption with many planted

relations. That is, we need to assume that the following distribution is indistinguishable from $\mathcal{D}_0^m$:

$\mathcal{D}_r^m$: a random m-dimensional linear subspace $V_{s_1,\ldots,s_r} \subseteq \mathbb{F}_2^n$ satisfying r random planted t-XOR relations $s_1, \ldots, s_r$ (i.e., $s_1, \ldots, s_r$ are random t-sparse vectors and $s_1 \cdot v = \cdots = s_r \cdot v = 0$ for all $v \in V_{s_1,\ldots,s_r}$).

We are not aware of any prior work on this assumption that $\mathcal{D}_0^m \approx \mathcal{D}_r^m$, so it is not immediately clear how reliable it is. Fortunately, it is *implied* by the $(n, m + r, t)$ planted XOR assumption.

We prove this with a hybrid argument. Suppose that an efficient adversary $\mathcal{A}$ distinguishes between $\mathcal{D}_0^m$ and $\mathcal{D}_r^m$ with advantage $\varepsilon > 0$. By a telescoping argument, $\mathcal{A}$ must distinguish between $\mathcal{D}_i^m$ and $\mathcal{D}_{i+1}^m$ with advantage ε/r, for some $i \in \{0, \ldots, r-1\}$. For each i, the following efficient reduction Red_i satisfies $\mathsf{Red}_i(\mathcal{D}_0^{m+r}) \equiv \mathcal{D}_i^m$ and $\mathsf{Red}_i(\mathcal{D}_1^{m+r}) \equiv \mathcal{D}_{i+1}^m$, which implies that ε/r (and therefore ε) is negligible under the $(n, m + r, t)$ planted XOR assumption.

$\mathsf{Red}_i(W)$:

1. Sample i random t-sparse vectors $s_1, \ldots, s_i \in \mathbb{F}_2^n$ and let $S = \{v \in \mathbb{F}_2^n : v \cdot s_j = 0 \ \forall j \in [i]\}$.
2. Let $U = W \cap S$. Notice that $\dim U \geq \dim W - i$. Since $\dim W = m + r$ and $i < r$, this is at least m.
3. Output a random m-dimensional subspace of U.

It remains to see why $\mathsf{Red}_i(\mathcal{D}_0^{m+r}) \equiv \mathcal{D}_i^m$ and $\mathsf{Red}_i(\mathcal{D}_1^{m+r}) \equiv \mathcal{D}_{i+1}^m$. In fact both of these statements are true even for *fixed* planted relations.

- $\mathsf{Red}_i(\mathcal{D}_0^{m+r}) \equiv \mathcal{D}_i^m$. Fix $s_1, \ldots, s_i$ sampled in Red_i and let $S = \{v \in \mathbb{F}_2^n : v \cdot s_j = 0 \ \forall j \in [i]\}$. Since W is a random subspace of $\mathbb{F}_2^n$, conditioned on any $d = \dim(W \cap S)$, $U = W \cap S$ is a random d-dimensional subspace of S. Therefore the output of $\mathsf{Red}_i(\mathcal{D}_0^{m+r})$ is a random m-dimensional subspace of S.
- $\mathsf{Red}_i(\mathcal{D}_1^{m+r}) \equiv \mathcal{D}_{i+1}^m$. Fix $s_1, \ldots, s_i$ sampled in Red_i and let $S = \{v \in \mathbb{F}_2^n : v \cdot s_j = 0 \ \forall j \in [i]\}$, as above. Suppose that $W \leftarrow \mathcal{D}_1^{m+r}$ is sampled with the planted relation s. Fix s and let $S' = \{v \in S : v \cdot s = 0\}$. Again, conditioned on any $d = \dim(W \cap S)$, $U = W \cap S = W \cap S'$ is a random d-dimensional subspace of S'. Therefore the output of $\mathsf{Red}_i(\mathcal{D}_1^{m+r})$ is a random m-dimensional subspace of S'.

Narrow, Statistically Random Generator Matrix. Since the planted XOR assumption is not a standard cryptographic assumption, we show that the generator matrix of $\mathsf{LDPC}[n, c\log^2 n, \log n, 0.99n]$ is *statistically* random for some $c > 0$. This removes the need for the planted XOR assumption, but it comes at the cost of requiring a stronger version of the LPN assumption: When we invoke LPN to see that samples $(G, Gs \oplus e)$ are pseudorandom, the secrets s are now only of size $c\log^2 n$. Therefore, for this PRC we will rely on a subexponential version of the LPN assumption which states that LPN is $2^{O(\sqrt{n})}$-hard.

For the purposes of this technical overview, we will show that the generator matrix of a closely related code is random. The proof for this version is significantly simpler, but the distribution is less natural and has worse error-correcting properties. The modified distribution on (P, G) is defined as follows:

1. Sample a uniformly random $G_0 \leftarrow \mathbb{F}_2^{0.01n \times g}$.
2. For $i \in [0.99n]$:
   (a) Sample a random $(t-1)$-sparse $s_i \in \mathbb{F}_2^{0.01n}$.
   (b) Let G_i be the matrix G_{i-1} with the extra row $s_i^T G_0$ appended to the bottom,
   $$G_i = \begin{bmatrix} G_{i-1} \\ s_i^T G_0 \end{bmatrix}.$$
   (c) Let $s'_i = [s_i^T, 0^{i-1}, 1, 0^{0.99n-i}]$.
3. Let P be the matrix whose rows are $s'_1, \ldots, s'_{0.99n}$ and $G = G_{0.99n}$. Output (P, G).

First observe that $PG = 0$, because $s'_i G = [s_i^T, 0^{i-1}, 1, 0^{0.99n-i}]G = s_i^T G_0 \oplus (s_i^T G_0) = 0$ for every $i \in [0.99n]$.

The leftover hash lemma immediately implies that G is statistically random. Recall that the leftover hash lemma states that if $A \leftarrow \mathbb{F}_2^{g \times \ell}$ is a uniformly random matrix and $s \in \mathbb{F}_2^{\ell}$ has min-entropy μ, then (A, As) is $2^{-(\mu-g)/2}$-close to uniform in statistical distance. In our case, we use $A = G_0^T$ and $s = s_i$ to see that $s_i^T G_0$ is $2^{-(\log \binom{0.01n}{t} - g)/2}$-close to uniform in statistical distance for each $i \in [n-g]$. If $t = \Omega(\log n)$, then there is a choice of $g = \Omega(\log^2 n)$ such that $2^{-(\log \binom{0.01n}{t} - g)/2} = 2^{-\Omega(\log^2 n)} = \mathsf{negl}(\lambda)[n]$, completing the proof.

$\mathsf{LDPC\text{-}PRC}_0$ *is robust to any p-bounded channel.* Recall that we say that any length-preserving channel that introduces at most a p fraction of bit-flip errors is p-bounded. To prove robustness, we need to show two things:

1. any fixed $x \in \mathbb{F}_2^n$ decodes to $\perp$ with high probability, and
2. for any p-bounded channel $\mathcal{E}$, samples from $\mathcal{E}(\mathsf{Encode}_G(1))$ decode to 1 with high probability.

Unfortunately, (1) does not quite hold for the scheme presented above. The issue is that, while *most* fixed strings will decode to $\perp$, a small fraction of strings will decode to 1 regardless of P. For instance, 0 will always decode to 1 because $\mathrm{wt}(P \cdot 0) = 0$ for any P. Therefore we modify our scheme slightly by using a one-time pad $z \in \mathbb{F}_2^n$, included in the public key. The modified $\mathsf{Encode}_G(1)$ outputs $Gs \oplus e \oplus z$, and $\mathsf{Decode}_P(x)$ computes $\mathrm{wt}(Px \oplus Pz)$ instead of $\mathrm{wt}(Px)$; this is the actual scheme we describe in the full version.

Now, for any fixed $x \in \mathbb{F}_2^n$, $\mathrm{wt}(Px \oplus Pz)$ is distributed identically to $\mathrm{wt}(Pz)$ because z is uniform. In the full version we show that P is full rank with high probability, so Pz is uniformly random. By a Chernoff bound, $\mathrm{wt}(Pz) \geq (1/2 - r^{-1/4}) \cdot r$ with high probability and therefore $\mathsf{Decode}_P(x)$ outputs $\perp$. This concludes the proof of (1), so we turn to (2).

Suppose that we sample $Gs \oplus e \oplus z \leftarrow \mathsf{Encode}_G(1)$ and apply some p-bounded channel $\mathcal{E}$. The one-time pad effectively converts $\mathcal{E}$ to a fixed error channel, independent of P, G, s, e: Suppose that $\mathcal{E}(x) = x \oplus e(x)$, where $e(x)$ is a random variable depending on x. Since $\mathcal{E}$ is p-bounded, $\mathrm{wt}(e(x)) \leq p \cdot n$. Letting $y = Gs \oplus e \oplus z$, we have

$$\begin{aligned}\mathcal{E}(Gs \oplus e \oplus z) \oplus z &= (Gs \oplus e \oplus z) \oplus e(Gs \oplus e \oplus z) \oplus z \\ &= Gs \oplus e \oplus e(y)\end{aligned}$$

where y is uniformly random in $\mathbb{F}_2^n$, independent of P, G, s, e because of z. Now it only remains to see that $\mathrm{wt}(P(Gs \oplus e \oplus e(y))) = \mathrm{wt}(P(e \oplus e(y))) < (1/2 - r^{-1/4}) \cdot r$ with high probability. Since e and $e(y)$ are independent errors, each of weight $(1/2 - \Omega(1)) \cdot n$, the combined error $e \oplus e(y)$ also has weight $(1/2 - \Omega(1)) \cdot n$. Therefore, if the row sparsity t of P is $c \log n$ for sufficiently small constant c, then we will have $\mathrm{wt}(P(e \oplus e(y))) < (1/2 - r^{-1/4}) \cdot r$ with high probability, completing the proof of (2).

2.3 Pseudorandom Codes for the Deletion Channel

So far, we have only considered PRCs for substitution channels. For our applications to watermarking and steganography, it will be useful to have PRCs for the noisy deletion channel as well. The noisy deletion channel randomly introduces both deletions and substitutions.

Unfortunately, existing error-correcting codes for the deletion channel introduce a large amount of structure into codewords that precludes pseudorandomness. For instance, the popular techniques of synchronization symbols or concatenation with constant-sized inner codes are immediately seen to be incompatible with pseudorandomness. Even further limiting the techniques available to us, we want our PRCs for the noisy deletion channel to have a binary alphabet in order to be useful for watermarking.

We therefore turn to alternative techniques. Surprisingly, we find that the repetition code—perhaps the simplest and most-structured error-correcting code—is a useful starting point.

For odd integer T, the rate-$(1/T)$ repetition code works by repeating each bit of the message T times. That is, for any message $\mathsf{m} = (\mathsf{m}_1 || \cdots || \mathsf{m}_k) \in \{0,1\}^k$, the encoder RepEnc_T is defined by $\mathsf{RepEnc}_T(\mathsf{m}) = (\mathsf{m}_1)^T || \cdots || (\mathsf{m}_k)^T$, where $(\mathsf{m}_i)^T$ denotes bit m_i repeated T times. For example, the rate-$(1/3)$ repetition code encodes 010 as $\mathsf{RepEnc}_T(010) = 000111000$.

Now suppose that the encoding $(\mathsf{m}_1)^T || \cdots || (\mathsf{m}_k)^T$ is subjected to the noisy deletion channel, resulting in a string z. A natural algorithm for decoding z is to partition z into k equal-length blocks $z_1, \ldots, z_k$, and compute the majority of each block:

$\mathsf{MajDec}_k(z)$:

1. Partition z into k equal-length blocks $z = (z_1 || \cdots || z_k)$.
2. Output $(\mathrm{Maj}(z_1) || \cdots || \mathrm{Maj}(z_k))$.

As long as the deletions are sufficiently balanced across the different blocks, the z_i will align well with the original blocks $(\mathsf{m}_i)^T$. Provided further that there are not too many substitutions in any block, we should have $\mathsf{MajDec}_k(z) = \mathsf{m}$. The issue is that $\mathsf{RepEnc}_T(\mathsf{m})$ is not pseudorandom even for random m, because a random string is (extremely) unlikely to consist of T repeated bits.

On the other hand, a random string typically does have $\Theta(\sqrt{T})$ bias towards 0 or 1.[5] So if we change or delete a small $O(\sqrt{T})$ number of bits of a random string, we expect the majority to stay the same. This observation brings us to the following encoder MajEnc_T, which encodes each bit in the majority of a random string. We refer to the code defined by $(\mathsf{MajEnc}_T, \mathsf{MajDec}_k)$ as the *majority code*.

$\mathsf{MajEnc}_T(\mathsf{m})$:

1. For $i \in [k]$, let z_i be a random sample from $\{0,1\}^T$ conditioned on $\mathrm{Maj}(z_i) = \mathsf{m}_i$.
2. Output $(z_1 || \cdots || z_k)$.

Now if m is random, then $z = \mathsf{MajEnc}_T(\mathsf{m})$ is random as well. Furthermore, if we subject z to the noisy deletion channel to obtain z', then the bits of $\mathsf{m}' = \mathsf{MajDec}_k(z')$ are positively correlated with the bits of m. This is because the deletions are at random locations, and are therefore (roughly) evenly-distributed across the different blocks z_i—meaning that MajDec_k will mostly use the correct locations to decode each bit. Since the bit-flip errors are random, they merely dilute the $\Theta(\sqrt{T})$ biases. If $T >> k$ and the rates of deletions and bit-flip errors are constants below 1 and 1/2 respectively, then we show in the full version that $\Pr[\mathsf{m}_i = \mathsf{m}'_i]$ is a constant greater than 1/2. Therefore, the majority code has the effect of converting the constant-rate noisy deletion channel into some p-bounded channel.

Of course, the majority code is not itself a PRC. The first problem is that codewords for the majority code are only random if the message is random, whereas a PRC needs to allow encoding of any particular message. The second problem is that, even if the message is random, the majority code recovers a string that is only *correlated* with it.

But these are exactly the problems solved by PRCs for bounded-weight error channels! That is, if PRC is any PRC with robustness to every p-bounded channel (e.g. the PRCs from Sect. 2.2), then the combined code $\mathsf{PRC}_{\mathsf{del}} = (\mathsf{MajEnc} \circ \mathsf{PRC.Encode}, \mathsf{PRC.Decode} \circ \mathsf{MajDec})$ is a PRC with robustness to some constant-rate noisy deletion channel.[6] Pseudorandomness follows from the pseudorandomness of $\mathsf{PRC.Encode}$: Since $\mathsf{PRC.Encode}(\mathsf{m})$ is pseudorandom for any message m, $\mathsf{MajEnc}(\mathsf{PRC.Encode}(\mathsf{m}))$ is as well. Robustness follows from the fact that the majority code has the effect of converting the constant-rate noisy deletion channel into some p-bounded channel, which is handled by $\mathsf{PRC.Decode}$.

[5] That is, a random string has $\Theta(\sqrt{T})$ more 0's than 1's, or 1's than 0's. This can be seen as a consequence of the fact that a one-dimensional simple random walk of length T will usually terminate $\Theta(\sqrt{T})$ away from the origin.

[6] As p approaches 1/2, the combined PRC tolerates a noisy deletion channel with rates of deletions and bit-flip errors approaching 1 and 1/2.

2.4 Constant-Rate Pseudorandom Codes

So far we have only considered zero-bit PRCs, but for many applications it will be useful to have PRCs that can encode longer messages. There is a simple construction of a multi-bit PRC directly from any zero-bit PRC: Encode each bit of the message with either a zero-bit PRC codeword, or a uniformly random string. That is, if PRC is a zero-bit PRC, we encode a message $\mathsf{m} \in \{0,1\}^k$ as $(x_1||\cdots||x_k)$ where for each $i \in [k]$, $x_i \leftarrow \{0,1\}^n$ if $\mathsf{m}_i = 0$ and $x_i \leftarrow \mathsf{PRC.Encode}(1)$ if $\mathsf{m}_i = 1$.

Unfortunately this scheme has a very low rate. If the zero-bit PRC has block length n, then the resulting multi-bit PRC has rate $1/n$. However, we show in the full version that one can use any such low-rate PRC to make any error-correcting code pseudorandom, with no asymptotic loss in rate.

The idea is to encode a seed for a one-time pad in the simple low-rate PRC just described, and then use the one-time pad to hide an error-correcting encoding of the message. More formally, let $(\mathsf{Enc}, \mathsf{Dec})$ be any (standard) error-correcting code and PRC be a low-rate PRC. We do not require $(\mathsf{Enc}, \mathsf{Dec})$ to have any cryptographic properties. We encode a message m as[7]

$$\mathsf{PRC.Encode}(r), \mathsf{Enc}(\mathsf{m}) \oplus \mathsf{PRG}(r),$$

where PRG is any pseudorandom generator and $r \leftarrow \{0,1\}^k$ is a fresh uniformly random string.

By pseudorandomness of PRC, $\mathsf{PRC.Encode}(r)$ is indistinguishable from a uniformly random string—even for a fixed choice of r. By security of PRG, the encoding is therefore indistinguishable from a totally random string.

Decoding works as long as $\mathsf{PRC.Encode}(r)$ is not too corrupted for PRC to recover r, and $\mathsf{Enc}(\mathsf{m}) \oplus \mathsf{PRG}(r)$ is not too corrupted for Dec to recover m.

2.5 Watermarks for Language Models

In this work, a generative language model is formally described by an algorithm Model that takes as input a prompt PROMPT and a sequence of tokens output thus far $\mathsf{t}_1, \ldots, \mathsf{t}_{i-1}$, and produces a distribution over the next token. A full response is generated by iteratively sampling from these distributions, at each step providing Model with the partial response, and terminating once a special "done" token is sampled. For simplicity, we assume here that tokens are binary, which allows us to specify the distribution p_i over the next token as a Bernoulli distribution $\mathrm{Ber}(\hat{\mathsf{p}}_i)$ where $\hat{\mathsf{p}}_i := \mathbb{E}[\mathsf{p}_i] \in [0,1]$. We also assume for the purposes of this technical overview that the model always generates a response of length n. In the full version we explain why neither of these assumptions is important for our results.

[7] In order to obtain stronger robustness guarantee, we actually randomly permute the symbols of this encoding. For the purposes of this technical overview we omit this detail.

As defined in [8], a watermarking scheme for a language model consists of algorithms Wat and Detect, where Wat is the watermarked model and Detect is an algorithm used to detect the presence of the watermark. In this work we are interested in watermarks that are *undetectable*, *sound*, and *robust*, loosely defined as follows.

- *Undetectability*: Any polynomial number of responses from the watermarked model are computationally indistinguishable from those of the original model.
- *Soundness*: Text generated independently of the watermarked model is not falsely detected.
- *Robustness*: Sufficiently high-entropy text output by the model is detected as watermarked, even if it is altered.

We show that the watermarking strategy from Sect. 1, which replaces some of the model's randomness with PRC codewords, yields a scheme that satisfies all of the above properties.

Defining Generate *for language models.* Recall that the approach from Sect. 1 requires an algorithm Generate that takes as input a prompt and a random seed $x \in \{0,1\}^n$, and samples a response $\mathsf{t} \in \{0,1\}^n$ such that

(1) if x is uniformly random, then t is distributed identically to a response from Model, and
(2) each bit of t is correlated with the corresponding bit of x.

We define $\mathsf{Generate}(\text{PROMPT}, x)$ to sample the i^{th} bit t_i of the response as follows. It first computes p_i by querying Model with PROMPT and the response output thus far, then:

- If $\hat{\mathsf{p}}_i \le 1/2$, sample $\mathsf{t}_i \leftarrow \mathrm{Ber}(2x_i\hat{\mathsf{p}}_i)$.
- If $\hat{\mathsf{p}}_i > 1/2$, sample $\mathsf{t}_i \leftarrow \mathrm{Ber}(1-2(1-x_i)(1-\hat{\mathsf{p}}_i))$.

For any $\hat{\mathsf{p}}_i \in [0,1]$, one can easily see that t_i is distributed as $\mathrm{Ber}(\hat{\mathsf{p}}_i)$ since x_i is a uniformly random bit. This means that Generate satisfies Condition (1) above.

For Condition (2), the bias toward the seed is stronger the closer $\hat{\mathsf{p}}_i$ is to $1/2$. It is strongest when $\hat{\mathsf{p}}_i = 1/2$ exactly, in which case t_i is sampled from $\mathrm{Ber}(x_i)$ and is therefore equal to x_i. At the other extreme, if $\hat{\mathsf{p}}_i = 0$ or 1, there is no bias. In general the response is a noisy version of the seed, where the amount of noise on the i^{th} token decays as the binary entropy of $\mathrm{Ber}(\hat{\mathsf{p}}_i)$ grows.

Replacing Seeds with PRC Codewords. We use PRC samples $x \leftarrow \mathsf{PRC.Encode}(1)$, instead of random samples $x \leftarrow \{0,1\}^n$, as the seeds in Generate. That is, if PRC is a zero-bit PRC, we let our watermarking scheme $\mathcal{W}[\mathsf{PRC}]$ be defined by

$\mathsf{Wat}(\text{PROMPT})$: Sample $x \leftarrow \mathsf{PRC.Encode}(1)$ and output a sample from $\mathsf{Generate}(\text{PROMPT}, x)$.
$\mathsf{Detect}(\mathsf{t})$: Compute $\mathsf{PRC.Decode}(\mathsf{t})$ and output the result.

By Condition (1) and the pseudorandomness property of PRC, the responses from Wat are computationally indistinguishable from those of the original model. By Condition (2) and the robustness property of PRC, the watermark will be detectable as long as the PRC is sufficiently powerful.

Remark. Depending on the kind of robustness of the PRC, substituting the entire seed with a single PRC sample results in a watermark that may or may not be detectable from just a subsequence. This is easily fixed by using $x = (x_1||\cdots||x_m)$, where $x_i \leftarrow \mathsf{PRC.Encode}(1)$ are independent PRC samples that are much shorter than the generated content. As long as the text contains *at least one* subsequence corresponding to a PRC sample x_i, the watermark will be detected.

PRC Error Correction and Watermark Robustness. To understand how error correction of PRC translates to robustness of $\mathcal{W}[\mathsf{PRC}]$, it is helpful to think of $\mathsf{Generate}$'s sampling process as a noisy *embedding channel* applied to the seed. That is, for a seed $x \in \{0,1\}^n$, let $\mathcal{E}_{\mathsf{Emb}}(x) = \mathsf{Generate}(\text{PROMPT}, x)$ be the "embedding channel" describing the noise in $x \mapsto \mathsf{t}$. For detection, it is sufficient for PRC to correct against the channel $\mathcal{E}_{\mathsf{Emb}}$, since watermarked responses are exactly samples from $\mathcal{E}_{\mathsf{Emb}}(x)$ for $x \leftarrow \mathsf{PRC.Encode}(1)$.

Robustness of $\mathcal{W}[\mathsf{PRC}]$ is determined by PRC's ability to correct from additional errors on top of $\mathcal{E}_{\mathsf{Emb}}$. Let $\mathcal{E}_{\mathsf{adv}}$ be a channel modeling the changes an adversary introduces to a watermarked response, so the overall error applied to t follows $\mathcal{E}_{\mathsf{adv}} \circ \mathcal{E}_{\mathsf{Emb}}$. If PRC is robust to $\mathcal{E}_{\mathsf{adv}} \circ \mathcal{E}_{\mathsf{Emb}}$, the watermark is robust to this adversary's modifications.

In the full version, we show that as long as the text has non-zero entropy, $\mathcal{E}_{\mathsf{Emb}}$ introduces errors at a rate of less than $1/2$. Therefore, using any PRC with robustness to every p-bounded channel, we immediately obtain watermarks that are robust to a constant rate of random substitutions.

Robustness of Our Watermark. Hashing-based watermarking schemes—including all existing undetectable schemes—are removable by the simple "emoji attack" [1,12]. In this attack, an adversary asks the model to respond to its prompt and insert an emoji between every word of its response. The adversary then deletes the emojis from the response. This attack removes any watermark that relies on the detector seeing contiguous sequences of watermarked text.

It turns out that hashing-based schemes [1,8,12] can easily be made robust to this particular attack.[8] However, if the adversary instead instructs the model to insert the emojis *randomly*, then we do not know how to make any hashing-based scheme robust. By constructing PRCs with robustness to random deletion channels, we give the first undetectable watermarking scheme that can resist this kind of attack.

In order to show this, we require a stronger assumption on the response: That $\mathcal{E}_{\mathsf{Emb}}$ behaves as the binary symmetric channel BSC_q for some $q \in (0, 1/2)$. The

[8] For instance, we could choose to only hash tokens whose index has the same parity as the token being sampled.

binary symmetric channel BSC_q is the channel that flips each bit of its input independently with probability q, so $\mathrm{BSC}_q(x) = x \oplus \mathrm{Ber}(q)$ for $x \in \{0,1\}$. Essentially, this is equivalent to the assumption that the response has high entropy *and does not repeat words too often.*

Under this assumption, if $\mathcal{E}_{\mathsf{adv}} = \mathrm{BDC}_p$ is the binary deletion channel for some $p \in (0,1)$, then we just need a PRC with robustness to $\mathcal{E}_{\mathsf{adv}} \circ \mathcal{E}_{\mathsf{Emb}} = \mathrm{BDC}_p \circ \mathrm{BSC}_q$. The binary deletion channel BDC_p is the channel that deletes each bit of its input independently with probability p. Indeed, we saw in Sect. 2.3 that there exist PRCs with robustness to $\mathrm{BDC}_p \circ \mathrm{BSC}_q$ for any $p \in (0,1), q \in (0,1/2)$.

2.6 Watermarks with Public Attribution

For the "standard" notions of watermarks considered so far, the goal is to determine whether a given text is a possibly-corrupted version of an output generated by a model. Standard watermarks are well-suited for applications such as detecting plagiarism, where one wishes to know if a model was used at all to produce a text, even if that text has been altered by the user.

A different use of watermarks is in *attributing* content to an LLM that generated it. For example, if harmful content generated by an LLM is found on social media, it would be useful to trace this content back to the model using the watermark. Ideally, anyone holding a *public* detection key should be able to trace the content. On the other hand, it should only be possible to embed the watermark by using a *secret* embedding key, in order to avoid falsely attributing text to any model.[9] In other words, to an attacker who does not know the secret key, the watermark should be *unforgeable.*

In addition to unforgeability, watermarks with public attribution have subtly different detection properties than standard watermarks. Robustness of standard watermarks means that an LLM-generated text will be detected even if small modifications are made. If robust watermarks were used for attribution, an attacker could use a model to generate a benign watermark text, then change a few words to make it offensive. By robustness, the watermark would still be present in this now-offensive content. So whereas robustness is a useful feature for standard watermarking, in the context of attribution it is actually an issue.

We therefore define a watermark with public attribution to have a separate detection algorithm called $\mathsf{AttrText}$, which is intentionally designed to *not* be robust. $\mathsf{AttrText}$, given a text x and a public detection key, indicates whether the model output verbatim a significant part of that text, and outputs that portion of the text if so.

In order to preserve the benefits of robust watermarking for applications like detecting plagiarism, our publicly attributable watermarks also retain a Detect algorithm (in addition to the $\mathsf{AttrText}$ algorithm) with the robustness of our

[9] Note that the roles of the public and secret keys are reversed here. For PRCs, a secret key is necessary for decoding, but anyone can encode with knowledge of a public key. Public-key PRCs are useful for public-key steganography.

standard watermarking schemes. One can choose at detection time whether one wants to use Detect for standard detection, or AttrText for attribution.

Our watermarking scheme with public attribution, $\mathcal{W}_{\mathsf{att}}[\mathsf{PRC}]$, is a natural extension of our regular watermarking scheme $\mathcal{W}[\mathsf{PRC}]$. Recall that $\mathcal{W}[\mathsf{PRC}]$ embeds a codeword of a zero-bit PRC into the model's response; the detector checks whether the given text is close to a codeword. Of course, if we use a PRC that encodes an arbitrary message (rather than only '1' as in a zero-bit PRC), then $\mathcal{W}[\mathsf{PRC}]$ will embed arbitrary messages in the text. $\mathcal{W}_{\mathsf{att}}[\mathsf{PRC}]$ does exactly this, where the message that it encodes is a *signature on the response output thus far*. AttrText decodes the given text to obtain this signature, and checks using the public detection key that it is a valid signature of a portion of the response. If so, this signed portion must have been generated by the model.

Concurrent work [10] also constructs a watermark with public detection, although this scheme is designed to have mild robustness (comparable to that of [8]) and therefore is not appropriate for attribution as-is. Their scheme can easily be modified to satisfy our definition of unforgeable public attribution, but it would then lose all robustness guarantees for standard watermarking. Our scheme simultaneously functions as a highly robust standard watermark via Detect, while also satisfying unforgeable public attribution via AttrText.

2.7 Robust Steganography

In steganography, the goal is to send a hidden message such that an observer cannot tell that a message is being sent at all. In the classic presentation, a prisoner wishes to secretly communicate with an outside party even though the warden is filtering their letters. If the warden detects any unusual language then the communication channel will be shut down, so the prisoner cannot simply encrypt the message: The warden should not only be unable to learn anything about the message, but should be unable to even detect that secret communication is occurring at all.

Steganography was formalized in [11]. In this presentation, there is some underlying *steganographic channel*,[10] a distribution with which the sender wishes to conceal a message. The sender is given sample access to this steganographic channel and sends a *stegotext* to the receiver. *Steganographic secrecy* requires that the distribution of stegotexts is indistinguishable from the steganographic channel, except to the receiver who can recover the message with a secret key.

[11] proves the security of a steganography scheme of [6] that can be constructed using any encryption scheme $(\mathsf{Encode}, \mathsf{Decode})$ with pseudorandom ciphertexts. The key idea behind this scheme is to embed each bit x_i of a pseudorandom encryption of the message by drawing a sample d_i from the steganographic channel such that $f(d_i) = x_i$ for some hash function f:
$\mathsf{Steg.Encode}(1, \mathsf{m})$ [6,11]:

[10] In the steganography literature this is usually just called a "channel"; we call it a "steganographic channel" to differentiate it from the coding-theoretic channels we use in the context of robustness.

1. Let $x = x_1||\ldots||x_n \leftarrow \mathsf{Encode}(1, \mathsf{m})$
2. For $i \in [n]$, sample a random d_i from the channel conditioned on $f(d_i) = x_i$
3. Output $d = d_1||\ldots||d_n$

The decoder $\mathsf{Steg.Decode}$ simply outputs $\mathsf{Decode}(1, f(d_1)||\ldots||f(d_n)) = \mathsf{Decode}(1, x) = \mathsf{m}$.

If x_i is uniform over $\{0, 1\}$, and f is perfectly unbiased for the channel, then d_i is sampled exactly from the channel distribution. Therefore, by pseudorandomness of the ciphertext, an observer cannot distinguish stegotexts from samples from the steganographic channel. The receiver, which knows the decoding key for the encryption scheme, can evaluate f on each block of the stegotext to obtain the ciphertext, then decrypt to recover the message.

However, this scheme is very brittle. If the stegotext is corrupted with any errors at all—even ones resulting from any small bias in the hash function f—the message cannot be recovered. A natural attempt at achieving robustness is for the sender to apply an error-correcting code $(\mathsf{Enc}, \mathsf{Dec})$ to the ciphertext before embedding it. But this loses pseudorandomness and therefore steganographic secrecy! Consequently, the robust steganography schemes of prior work rely on stronger assumptions like the ability of the sender and receiver to share state. A shared state allows the sender to generate a fresh one-time pad r for each message it sends, making the task much easier: The sender embeds $x = \mathsf{Enc}(\mathsf{m}) \oplus r$ by choosing d such that $f(d_i) = x_i$, and the receiver computes $\mathsf{Dec}(\tilde{x} \oplus r)$, where $(\mathsf{Enc}, \mathsf{Dec})$ is any error-correcting code. However, if the sender and receiver become unsynchronized (as is likely in practice), the receiver can no longer decode the message.

We observe that PRCs are exactly the primitive needed for robust stateless steganography: Using a PRC as the pseudorandom encryption scheme in the above $(\mathsf{Steg.Encode}, \mathsf{Steg.Decode})$ construction *immediately* gives us a steganography scheme with the same robustness as the PRC. If we use a public-key PRC, the resulting steganography scheme is also public-key. Furthermore, the robustness of the PRC allows us to relax the assumption that f is perfectly unbiased on the steganographic channel.

Our main result about steganography is the first stateless steganography scheme with nontrivial robustness to errors. In particular, using our LDPC-based PRCs, we construct stateless steganography schemes that are robust to p-bounded channels for any constant p, or any constant-rate random deletion channel.

References

1. Aaronson, S.: My AI Safety Lecture for UT Effective Altruism (2022). https://scottaaronson.blog/?p=6823. Accessed May 2023
2. Agrawal, S., Saha, S., Schwartzbach, N.I., Vanukuri, A., Vasudevan, P.N.: k-sum in the sparse regime. Cryptology ePrint Archive, Paper 2023/488 (2023). https://eprint.iacr.org/2023/488

3. von Ahn, L., Hopper, N.J.: Public-key steganography. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 323–341. Springer, Cham (2004). https://doi.org/10.1007/978-3-540-24676-3_20
4. Alekhnovich, M.: More on average case vs approximation complexity. In: Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS 2003), Cambridge, MA, USA, 11–14 October 2003, pp. 298–307. IEEE Computer Society (2003). https://doi.org/10.1109/SFCS.2003.1238204
5. Alon, N., Bruck, J., Naor, J., Naor, M., Roth, R.M.: Construction of asymptotically good low-rate error-correcting codes through pseudo-random graphs. IEEE Trans. Inf. Theory **38**(2), 509–516 (1992). https://doi.org/10.1109/18.119713
6. Anderson, R.J., Petitcolas, F.A.: On the limits of steganography. IEEE J. Sel. Areas Commun. **16**(4), 474–481 (1998)
7. Bartz, D., Hu, K.: OpenAI, Google, others pledge to watermark AI content for safety, White House says (2023). https://www.reuters.com/technology/openai-google-others-pledge-watermark-ai-content-safety-white-house-2023-07-21
8. Christ, M., Gunn, S., Zamir, O.: Undetectable watermarks for language models. IACR Cryptology ePrint Archive, p. 763 (2023). https://eprint.iacr.org/2023/763
9. Dodis, Y., Ganesh, C., Golovnev, A., Juels, A., Ristenpart, T.: A formal treatment of backdoored pseudorandom generators. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 101–126. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46800-5_5
10. Fairoze, J., Garg, S., Jha, S., Mahloujifar, S., Mahmoody, M., Wang, M.: Publicly detectable watermarking for language models. Cryptology ePrint Archive (2023)
11. Hopper, N.J., Langford, J., von Ahn, L.: Provably secure steganography. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 77–92. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45708-9_6
12. Kirchenbauer, J., Geiping, J., Wen, Y., Katz, J., Miers, I., Goldstein, T.: A watermark for large language models. arXiv preprint arXiv:2301.10226 (2023)
13. Kuditipudi, R., Thickstun, J., Hashimoto, T., Liang, P.: Robust distortion-free watermarks for language models. arXiv preprint arXiv:2307.15593 (2023)
14. Naor, J., Naor, M.: Small-bias probability spaces: efficient constructions and applications. SIAM J. Comput. **22**(4), 838–856 (1993). https://doi.org/10.1137/0222053
15. Rogaway, P., Bellare, M., Black, J.: OCB: a block-cipher mode of operation for efficient authenticated encryption. ACM Trans. Inf. Syst. Secur. (TISSEC) **6**(3), 365–403 (2003)
16. Ta-Shma, A.: Explicit, almost optimal, epsilon-balanced codes. In: Hatami, H., McKenzie, P., King, V. (eds.) Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2017, Montreal, QC, Canada, 19–23 June 2017, pp. 238–251. ACM (2017). https://doi.org/10.1145/3055399.3055408
17. Vazirani, U.V., Vazirani, V.V.: Trapdoor pseudo-random number generators, with applications to protocol design. In: 24th Annual Symposium on Foundations of Computer Science (SFCS 1983), pp. 23–30 (1983). https://doi.org/10.1109/SFCS.1983.78
18. Zhang, H., Edelman, B.L., Francati, D., Venturi, D., Ateniese, G., Barak, B.: Watermarks in the sand: impossibility of strong watermarking for generative models. arXiv preprint arXiv:2311.04378 (2023)
19. Zhao, X., Ananth, P., Li, L., Wang, Y.X.: Provable robust watermarking for AI-generated text. arXiv preprint arXiv:2306.17439 (2023)