

Cryptanalysis of LDPC-Based Pseudorandom Error-Correcting Codes

Tianrui Wang[1] Anyu Wang[2,3,4†] Tianshuo Cong[5,6†]
Delong Ran[1] Jinyuan Liu[2] Xiaoyun Wang[2,3,4,7,8]
[1]*Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University*
[2]*Institute for Advanced Study, Tsinghua University* [3]*Zhongguancun Laboratory, Beijing, China*
[4]*State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University*
[5]*School of Cryptologic Science and Engineering, Shandong University*
[6]*Shandong Key Laboratory of Artificial Intelligence Security, Shandong University*
[7]*National Financial Cryptography Research Center, Beijing, China*
[8]*Shandong Institute of Blockchain, Shandong, China*

Abstract

Pseudorandom error-correcting codes (PRCs), a novel cryptographic primitive recently proposed at CRYPTO 2024, are primarily applied in undetectable watermarking schemes for large generative models. However, the security of PRCs has not yet been systematically analyzed. To fill this gap, we present the first cryptanalysis of PRCs. Specifically, focusing on LDPC-PRC, the only known practical instantiation of PRCs, we propose three novel attacks that challenge its undetectability and robustness. To rigorously demonstrate the practical threat, we analyze the concrete attack complexity under realistic parameters and validate the attack effectiveness on both real-world large language models and generative image models, including DeepSeek and Stable Diffusion. Our analysis shows that the claimed security guarantees of LDPC-PRC are undermined across all practically feasible regimes. For example, our attacks can detect the presence of a watermark with overwhelming probability at a cost of 2^{22} operations. Beyond attacks, we further propose three defenses: parameter recommendation, implementation suggestion, and a revised key generation function. However, PRC-based watermarking schemes still fail to achieve 128-bit security due to inherent constraints of large generative models, such as the maximum output length of large language models. Overall, our work clarifies the concrete security limits of PRCs in real-world watermarking applications.

† Corresponding authors: Anyu Wang (anyuwang@tsinghua.edu.cn) and Tianshuo Cong (tianshuo.cong@sdu.edu.cn)

1 Introduction

The rapid growth of generative Artificial Intelligence (AI) models, such as GPT-4o [34] and Stable Diffusion [38], has made AI-Generated Content (AIGC) ubiquitous [18, 37]. However, this widespread adoption is accompanied by substantial misuse risks [2, 27]. Consequently, reliable identification and provenance tracking of AIGC have become critical for AI governance, with content watermarking standing out as one of the most promising technical solutions [49].

Notably, a compelling foundation for AIGC watermarks is the recently developed cryptographic primitive of Pseudorandom Error-correcting Codes (PRCs) [8]. A PRC scheme comprises three algorithms $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$. The KeyGen algorithm generates a public-secret key pair. Any party can use the public key with Encode to embed a message into a codeword, while only the holder of the corresponding secret key can apply Decode to perform verification. Crucially, PRCs are defined by two core security properties: *robustness*, which guarantees that a valid codeword, even after limited distortion, will decode to the original message; and *undetectability*, which ensures that codewords are computationally indistinguishable from random vectors without the secret key. These properties capture the essential requirements for a practical watermark by ensuring it resists removal and detection by adversaries, making PRCs particularly suitable for constructing stealthy and robust watermarks for both Large Language Models (LLMs) and Generative Image Models (GIMs).

To instantiate these properties, Christ and Gunn proposed LDPC-PRC [8], an instantiation of PRCs based on Low-Density Parity-Check (LDPC) codes. LDPC-PRC offers provable security under standard assumptions, namely the hardness of the Learning Parity with Noise (LPN) and planted XOR problems. These theoretical foundations provide PRC-

based watermarking shemes with significant advantages over other heuristic alternatives [1, 9, 12, 21, 24, 48].

Despite strong theoretical guarantees, the concrete security of PRCs under practical parameter settings remains largely unexplored. In particular, existing PRCs including LDPC-PRC are vulnerable to quasipolynomial-time distinguishing attacks [7], indicating that their practical undetectability may be significantly weaker than expected. Although undetectability was originally introduced to preserve the quality of generated content, in realistic watermarking scenarios, breaking PRC undetectability can directly lead to unauthorized watermark detection, information leakage, or further facilitate subsequent watermark removal and forgery attacks. Likewise, compromising PRC robustness directly leads to targeted watermark removal attacks. Accordingly, a rigorous security evaluation of PRCs under practical parameters is urgently needed.

1.1 Our Contributions

This work presents the first concrete security analysis of PRCs, bridging the gap between asymptotic security guarantees and practical deployment requirements. Since LDPC-PRC is the only feasible instantiation of PRC to date, our analysis focuses on designing attacks to challenge the undetectability and robustness of LDPC-PRC, consequently facilitating the process of detecting and removing the watermarks of LDPC-PRC-based watermarking schemes.

Novel Cryptographic Attacks. We propose three attacks against LDPC-PRC, with goals ranging from watermark detection to watermark removal. Table 1 summarizes the theoretical complexity of each attack.

- **Attack-I: Partial Secret Key Recovery:** This attack aims to recover part of the secret key from the public key, enabling an adversary to emulate the Decode() procedure of LDPC-PRC. This gives the adversary a constant distinguishing advantage greater than $\frac{1}{2}$ between codeword and random vector, allowing watermark detection with overwhelming probability given sufficient samples.
- **Attack-II: Weak Key Distinguisher:** This attack exploits an implementation-level vulnerability in the key generation procedure. We observe that in instantiated LDPC-PRC-based watermarking schemes, including those for LLMs [8] and GIMs [16], public keys may exhibit statistical non-uniformity with non-negligible probability. We demonstrate that an adversary can leverage this non-uniformity to detect the presence of a watermark with overwhelming probability.
- **Attack-III: Noise Overlay Attack:** This attack exploits the bounded noise tolerance of the LDPC-PRC's Decode algorithm. By recovering the original Bernoulli noise vector $\mathbf{e}$, an adversary can construct a carefully designed overlay vector $\mathbf{e}'$ of a fixed noise weight such that $\mathbf{e}+\mathbf{e}'$ exceeds the decoder's correction threshold, whereas a random $\mathbf{e}'$ with the same weight would not. This results in watermark removal without violating the practical constraints of LLM or GIM applications.

Table 1: The summary of the attack complexities. n is the code length of LDPC-PRC, g is the column dimension of the public key $\mathbf{G}$, t is the row weight of the secret key $\mathbf{P}$. The α in Attack-I is calculated in Theorem 1. For Attack-II, P is the probability that a weak key exists. The ε in Attack-III corresponds to the error-correcting capability.

	Goal	Time Complexity
Attack-I	Against Undetectability (Watermark Detection)	$g \cdot \alpha \cdot \binom{n/2}{\lceil t/2 \rceil}$ Theorem 1
Attack-II	Against Undetectability (Watermark Detection)	$\mathrm{P}^{-1} \cdot n \cdot g$ Theorem 2
Attack-III	Against Robustness (Watermark Removal)	$(\frac{1}{2}+\varepsilon)^{-g} \cdot n^3$ Theorem 3

Complexity for Concrete Parameter Configurations. We further analyze the concrete complexities of our attacks by incorporating specific parameter settings for LDPC-PRC-based LLM watermarking scheme (denoted as Π_{LLM} [8]) and GIM watermarking scheme (denoted as Π_{GIM} [16]). For Attack-I, our analysis shows that the complexities are sub-security-level for all parameter choices, where the actual security of Π_{GIM} remains *below* 46 *bits*. Attack-II further exposes vulnerabilities resulting from a non-negligible frequency of weak keys across all parameter choices. For example, weak keys appear with a nearly 100% probability when $t = 3, 4$ for Π_{LLM}, under which a distinguisher can be constructed in polynomial time. For Attack-III, we prove that the security level of Π_{LLM} *cannot exceed* 75 *bits* for any parameter choice with $n \leq 2^{17}$ under the threat of noise overlay attack. Meanwhile, the security level of Π_{GIM} *is below* 60 *bits* with a much lower concrete noise rate.

Evaluation on Real-World Applications. We choose DeepSeek [17] and Stable Diffusion [38] as real-world LLM and GIM to implement Π_{LLM} and Π_{GIM}, respectively. To the best of our knowledge, this is the first real-world implementation for Π_{LLM}. However, we validate that Π_{LLM} is impractical for embedding watermarks into LLMs as the entropy of output tokens is too low under reasonable text qualities, rendering the watermark undetectable by the original decoder. Furthermore, evaluation results demonstrate that both the proposed Attack-I and Attack-II can achieve a 100% attack success rate against Π_{LLM} and Π_{GIM} under certain parameter choices (refer to Table 3 and Table 4).

Mitigation Methods. We discuss strategies to mitigate the

threats posed by the proposed attacks. For Attack-I, we suggest that selecting appropriate parameters is sufficient to make recovering any part of the secret key computationally difficult. For Attack-II, we propose a revised key generation algorithm that significantly reduces the probability of generating weak keys while maintaining the randomness of the LDPC-PRC keys. Regarding Attack-III, the most effective defense is to increase the code length n to raise the attack's complexity. However, achieving a high security level, e.g., 128-bit, may necessitate an unreasonably large n. Our analysis shows that $n > 2^{24}$ is a necessary condition for 128-bit security across all choices of t. This requirement far exceeds the maximum token length of state-of-the-art LLMs (only 2^{15}). Consequently, defending against Attack-III through parameter selection alone remains difficult for practical LLM implementation. Additionally, We further suggest a modification to the GIM implementation, which mitigates the vulnerability caused by its low-weight noise parameter.

Takeaways. Our analysis reveals a practical limitation of current PRC-based LLM watermarking schemes, where preventing practical watermark-removal attacks requires an impractically large code length n, far beyond the output length supported by current LLMs. In contrast, PRC-based image watermarking may still admit practical instantiations with acceptable perceptual quality, although their undetectability guarantees can be weakened in practice.

2 Related Works

Theoretical Studies on PRCs. Due to the promising potential of PRCs, besides studying how to apply PRCs to AIGC watermarking [8, 16], several fundamental theoretical studies on PRCs have recently emerged, such as proving CCA-security based on the hardness of LPN [3], discussing unconditionally indistinguishability against space-bounded adversaries [15], and performing a black-box reduction from one-way functions with sub-constant error rates [14]. Unfortunately, these studies have not thoroughly explored the security of PRCs. Note that although [3, 8] provide a theoretical evaluation of PRC's robustness, it does not conduct a security assessment in conjunction with the concrete parameters.

Watermarking Schemes for AIGC. AIGC watermarks can typically be classified into two categories: in-processing schemes and post-processing schemes. Post-processing schemes embed imperceptible yet detectable signals after text generation [25,40] or image generation [10,44,50], inevitably resulting in content quality degradation. Instead, both Π_{LLM} and Π_{GIM} belong to the in-processing approach [13,22,46–48], wherein watermark insertion is merged with the process of generating content. With a pseudorandom codeword embedded, the distribution of the watermarked output texts of Π_{LLM} is proven to be identical to that of the original texts, while Π_{GIM} utilizes a watermarked latent that maintains a Gaussian distribution to ensure watermark invisibility.

Attacks against AIGC Watermarks. To remove watermarks, mainstream methods typically apply heuristic perturbations to the AIGC, such as paraphrasing [23,35] or inserting perturbations [20, 22] for the generated texts, or using processing operations [4, 43] such as photometric distortions for images. Unlike these empirical removal attacks, our cryptographic attacks provide provable success rates and well-defined complexity. For PRC watermark detection, Gunn et al. [16] trained a ResNet-18 classifier [19] and reported that it fails to distinguish LDPC-PRC-based watermarked images from non-watermarked ones. However, we propose two different attacks to challenge the undetectability of Π_{GIM}.

3 Preliminaries

3.1 Pseudorandom Error-correcting Codes

We begin by introducing the formal definition of PRC and its instantiation based on LDPC codes. Our focus is on public-key PRCs due to their broader applicability, as they allow any party to embed a watermark while reserving detection capability for a regulator holding the secret key. In contrast, secret-key PRCs enable any party capable of embedding a watermark to also detect it locally, which makes forgery or removal attacks easier for malicious users.

3.1.1 Definition of PRCs

Definition 1 (Zero-bit public-key PRCs [8])**.** *Let Σ be a fixed alphabet, $\mathcal{E} : \Sigma^* \to \Sigma^*$ be a noisy channel modeled by adding a noise vector with a Bernoulli distribution, and $k(\lambda) \in \mathbb{N}$ be the length of the message under any fixed λ. A PRC is a triple of polynomial-time randomized algorithms $(\mathsf{KeyGen}, \mathsf{Encode}, \mathsf{Decode})$ that satisfy*

- *(**Robustness**) For any message $\mathsf{m} \in \Sigma^{k(\lambda)}$ and any $\lambda \in \mathbb{N}$, let $\mathbf{x} \leftarrow \mathsf{Encode}(1^\lambda, \mathsf{pk}, \mathsf{m})$, $\mathbf{x}' \leftarrow \mathcal{E}(\mathbf{x})$ then*

$$\Pr_{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)} [\mathsf{Decode}(1^\lambda, \mathsf{sk}, \mathbf{x}') = \mathsf{m}] \geq 1 - \mathsf{negl}(\lambda).$$

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

$$\Pr_{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)} [\mathsf{Decode}(1^\lambda, \mathsf{sk}, \mathbf{c}) = \perp] \geq 1 - \mathsf{negl}(\lambda).$$

- *(**Undetectability**) For any polynomial-time adversary $\mathcal{A}$, let $\Pr[\mathsf{E}] = \Pr[\mathcal{A}^{\mathsf{Encode}(1^\lambda, \mathsf{pk}, \cdot)}(1^\lambda, \mathsf{pk}) = 1]$ and $\Pr[\mathsf{U}] = \Pr[\mathcal{A}^{\mathcal{U}}(1^\lambda, \mathsf{pk}) = 1]$, then*

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

where $\mathcal{A}^{\mathcal{U}}$ denotes an adversary with access to a uniform random oracle along with previous queries.

Definition 1 describes zero-bit PRCs, where the encoded message m is a single fixed message and can therefore be omitted in the following discussions. Multi-bit PRCs are constructed upon this zero-bit primitive [8]. In this paper, we discuss attacks specifically against zero-bit PRCs, and these attacks naturally extend to the multi-bit setting. Furthermore, as noted in [8], the soundness property is primarily a technical condition ensuring non-triviality. Consequently, our attacks do not target this property.

Relationship to Stronger Robustness Definitions. In this work, we focus on the basic robustness notion in Definition 1, where the adversary perturbs through a substitution noise. Other stronger robustness definitions have also been considered. For example, adaptive robustness in [3] grants the adversary interactive query access of the decoder. Substring robustness in [8] further allows deletion-based cropping attacks in addition to the substitution perturbations already covered by Definition 1. Therefore, these robustness notions assume strictly stronger adversarial capabilities.

Our attack in Section 4.3 constructs perturbations by exploiting the underlying error structure of the scheme. Since adaptive robustness and substring robustness only extend the adversarial capabilities beyond those in Definition 1, the same attack strategy can be directly applied to break these stronger robustness notions as well.

3.1.2 Instantiation of PRCs

Construction 1 (LDPC-PRC [8])**.** *Based on LDPC codes, an instantiation of PRC named* LDPC-PRC *can be constructed as follows.*

- $\mathsf{KeyGen}(1^\lambda)$:
  1. *Sample a random matrix* $\mathbf{P} \in \mathbb{F}_2^{r\times n}$ *where each row of* $\mathbf{P}$ *has weight* t.
  2. *Sample a random matrix* $\mathbf{G} \in \mathbb{F}_2^{n\times g}$ *where* $\mathbf{PG} = \mathbf{0}$.
  3. *Sample a random vector* $\mathbf{z} \in \mathbb{F}_2^n$.
  4. *Output public key* $\mathsf{pk} = (\mathbf{G}, \mathbf{z})$ *and secret key* $\mathsf{sk} = (\mathbf{P}, \mathbf{z})$.
- $\mathsf{Encode}(1^\lambda, (\mathbf{G}, \mathbf{z}))$:
  1. *Sample a random vector* $\mathbf{s} \in \mathbb{F}_2^g$ *and* $\mathbf{e} \leftarrow \mathrm{Ber}(n, \omega)$.
  2. *Output the codeword* $\mathbf{x} \leftarrow \mathbf{Gs} + \mathbf{e} + \mathbf{z}$.
- $\mathsf{Decode}(1^\lambda, (\mathbf{P}, \mathbf{z}), \mathbf{x})$: *If* $w_H(\mathbf{P}(\mathbf{x}+\mathbf{z})) \le (\frac{1}{2} - r^{-1/4}) \cdot r$, *output 1; otherwise output* $\perp$.

For clarity, we list the key mathematical notations for LDPC-PRC in Table 5 (Appendix A).

As defined in Construction 1, the LDPC-PRC scheme begins by generating a key pair. The secret key $\mathbf{P}$ is an LDPC parity-check matrix, while the public key $\mathbf{G}$ is a matrix whose column space is the dual subspace of $\mathbf{P}$. The random vector $\mathbf{z}$ serves as a one-time pad. Without it, the zero vector would always decode to 1, violating the soundness property.

Under the planted XOR assumption, the public key $\mathbf{G}$ does not leak information about the secret key $\mathbf{P}$. Furthermore, the hardness of the LPN problem ensures that the encoded message $\mathbf{Gs} + \mathbf{e}$ is computationally indistinguishable from a uniform random vector.

During decoding, the Hamming weight of $\mathbf{P}(\mathbf{x}+\mathbf{z}) = \mathbf{P}(\mathbf{Gs}+\mathbf{e}) = \mathbf{Pe}$ is significantly lower than $\frac{r}{2}$ due to the sparsity of both $\mathbf{P}$ and $\mathbf{e}$. In contrast, the product of $\mathbf{P}$ with a random vector yields a Hamming weight close to $\frac{r}{2}$. In the zero-bit setting, this decoding procedure is equivalent to watermark detection. Consequently, the watermark can be detected with overwhelmingly high probability by comparing the Hamming weight to an appropriately chosen threshold.

3.2 Large Generative Models

As PRCs are primarily applied in undetectable watermarking schemes for large generative models, this part introduces the fundamental generative principles of such models to facilitate a more comprehensive understanding of watermarking schemes. The formal definitions of LLMs and GIMs are as follows.

Definition 2 (LLM)**.** *A large language model* LLM *over a token set* $\mathcal{T}$ *is a probabilistic algorithm that takes a prompt* $\mathsf{prompt} \in \mathcal{T}^*$ *as input and generates succeeding text* $T \in \mathcal{T}^{\mathsf{len}}$. *The generation process of* LLM *is detailed in Algorithm 3.*

Definition 3 (GIM)**.** *A generative image model* GIM *is a probabilistic algorithm that takes a prompt* $\mathsf{prompt} \in \mathcal{T}^*$ *as input and outputs an image* $\mathsf{Img} \in [0,255]^{c\times w\times h}$. *The generation process of* GIM *is detailed in Algorithm 4.*

For LLMs, the model takes a prompt as input and generates text in a token-by-token manner. At each step, the model receives the initial prompt combined with the previously generated tokens as input, producing a probability distribution over the vocabulary from which the subsequent token is sampled accordingly. Watermarks can be embedded by influencing this probability distribution using a vector generated by LDPC-PRC.

For GIMs, the model first samples an initial latent from a Gaussian distribution and then progressively denoises it using a `U-net` conditioned on the input prompt. Finally, an image is generated via an autoencoder decoder `D-net`. Watermarks are embedded by selecting the signs of the initial latent according to the binary string generated by LDPC-PRC.

3.3 PRC-Based Watermarking Schemes

We formalize the key properties of a watermarking scheme for generative models in Definition 4. Informally, *robustness* captures the difficulty of removing the watermark under bounded

Algorithm 1 Key Generation Algorithm of Π_{LLM} [8]

Input: Parameters n, $r = 0.99n$, g, and t
Output: Matrices $\mathbf{P}$ and $\mathbf{G}$ for watermark generation

1: Sample a uniformly random matrix $\mathbf{G_0} \leftarrow \mathbb{F}_2^{0.01n \times g}$
2: Initialize an empty list for rows of $\mathbf{P}$
3: **for all** $i \in [0.99n]$ **do**
4: Sample a random $(t-1)$-sparse vector $\mathbf{s}_i \in \mathbb{F}_2^{0.01n}$
5: Compute $\mathbf{G}_i$ by appending the row $\mathbf{s}_i^T \mathbf{G}_0$ to $\mathbf{G}_{i-1}$:

$$\mathbf{G}_i = \begin{bmatrix} \mathbf{G}_{i-1} \\ \mathbf{s}_i^T \mathbf{G}_0 \end{bmatrix}$$

6: Construct $\mathbf{s}'_i = [\mathbf{s}_i^T, 0^{i-1}, 1, 0^{0.99n-i}]$
7: Append $\mathbf{s}'_i$ to the list of rows for $\mathbf{P}$
8: **end for**
9: Let $\mathbf{P}$ be the matrix whose rows are $\mathbf{s}'_1, \ldots, \mathbf{s}'_{0.99n}$
10: Let $\mathbf{G} = \mathbf{G}_{0.99n}$
11: Sample a random permutation $\Pi \in \mathbb{F}_2^{n \times n}$ and let $\mathbf{P} \leftarrow \mathbf{P}\Pi^{-1}, \mathbf{G} \leftarrow \Pi\mathbf{G}$
12: **return** $(\mathbf{P}, \mathbf{G})$

disturbance, *soundness* requires a negligible false positive rate, and *undetectability* means that no efficient adversary can detect the presence of a watermark with non-negligible advantage.

Definition 4 (Watermarking Scheme)**.** *Let Σ be a fixed alphabet and $\mathcal{E} : \Sigma^* \to \Sigma^*$ be a channel with the security parameter λ, a watermarking scheme is a tuple of polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{Wat}, \mathsf{Detect})$ *that satisfy*

- *(Robustness) For any content $x \leftarrow \mathsf{Wat}_{\mathsf{pk}}(\mathsf{prompt})$,*

$$\Pr_{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)} [\mathsf{Detect}_{\mathsf{sk}}(\mathcal{E}(x)) = \mathsf{true}] \geq 1 - \mathsf{negl}(\lambda).$$

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

$$\Pr_{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)} [\mathsf{Detect}_{\mathsf{sk}}(c) = \mathsf{false}] \geq 1 - \mathsf{negl}(\lambda).$$

- *(Undetectability) For any polynomial-time adversary $\mathcal{A}$ mimics* Detect*, let* Uni *be an oracle that outputs random vector in Σ^*, and $x \xleftarrow{\$} \mathsf{Uni}()$ with probability $\frac{1}{2}$ while $x \xleftarrow{\$} \mathsf{Wat}_{\mathsf{pk}}(\mathsf{prompt})$ with probability $\frac{1}{2}$, then*

$$\left| \Pr_{(\mathsf{sk},\mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda)} [\mathsf{Detect}_{\mathsf{sk}}(x) = \mathcal{A}_{\mathsf{pk}}(x)] - \frac{1}{2} \right| \leq \mathsf{negl}(\lambda),$$

3.3.1 Watermarking Scheme for LLMs

In [8], Christ and Gunn proposed an approach to design a watermarking scheme for LLMs (denoted as Π_{LLM}). The specific construction is presented in Construction 2.

Construction 2 (Π_{LLM} [8])**.** *A watermarking scheme Π_{LLM} for* LLM *over $\mathcal{T}$ is a tuple of polynomial-time algorithms $\Pi_{\text{LLM}} =$*$(\mathsf{Setup}, \mathsf{Wat}, \mathsf{Detect})$ *where*

- $\mathsf{Setup}(1^\lambda)$ *outputs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{LDPC\text{-}PRC.KeyGen}(1^\lambda)$. *The matrices are sampled through Algorithm 1.*
- $\mathsf{Wat}_{\mathsf{pk}}(\mathsf{prompt})$ *is a randomized algorithm that generates watermarked responses. The entire response generation procedure is similar to Algorithm 3, except for the token sampling process (Line 5). Specifically, Π_{LLM} first outputs a codeword $\mathbf{x} \leftarrow \mathsf{LDPC\text{-}PRC.Encode}(1^\lambda, \mathsf{pk})$, where $|\mathbf{x}| = \mathsf{len} * \lceil \log_2 |\mathcal{T}| \rceil$. Then, this codeword is split as* len *trunks and guides the token sampling process in a bit-wise manner through Algorithm 5.*
- $\mathsf{Detect}_{\mathsf{sk}}(T)$ *is an algorithm that outputs* true *or* false*. Π_{LLM} first extracts the codeword $\mathbf{x}'$ following Algorithm 6, then runs $\mathsf{LDPC\text{-}PRC.Decode}(1^\lambda, \mathsf{sk}, \mathbf{x}')$. If the result is* 1*, this algorithm outputs* true*, indicating the watermark is present; otherwise outputs* false*.*

Overall, Π_{LLM} embeds an LDPC-PRC codeword into the output text by biasing the token sampling probabilities according to the codeword bits. During generation, the codeword is partitioned into fixed-length segments, and each segment influences the sampling distribution of one token at the bit level. On the detection side, the embedded bits are reconstructed from the generated text and verified by applying the LDPC-PRC decoding algorithm with the secret key. However, Construction 2 is only a theoretical construction. In Section 6.2, we will analyze its practical utility and show that the output text quality is severely compromised, since the error-correcting capability of Π_{LLM} is too limited to support reliable detection under low token entropy. We further provide the first implementation of Π_{LLM} on a real-world LLM, empirically demonstrating its impracticality.

3.3.2 Watermarking Scheme for GIMs

Gunn et al. [16] further extended LDPC-PRC to the design of GIM watermarking schemes (denoted as Π_{GIM}). A detailed description of Π_{GIM} is provided in Construction 3.

Construction 3 (Π_{GIM} [16])**.** *A watermarking scheme Π_{GIM} for* GIM *is a tuple of polynomial-time algorithms* $(\mathsf{Setup}, \mathsf{Wat}, \mathsf{Detect}, \mathsf{Decode}')$ *where*

- $\mathsf{Setup}(1^\lambda)$ *outputs* $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{LDPC\text{-}PRC.KeyGen}(1^\lambda)$. *The matrices are sampled in a way similar to Algorithm 1, which will be discussed later.*
- $\mathsf{Wat}_{\mathsf{pk}}(\mathsf{prompt})$ *is a randomized algorithm that outputs a watermarked image* $\widetilde{\mathsf{Img}}$*. To this end, Π_{GIM} uses a watermarked latent $\tilde{\mathbf{y}}$ as $\mathbf{y}^s$ of Algorithm 4 to generate images. The generation process of $\tilde{\mathbf{y}}$ is*

$$\tilde{\mathbf{y}} = (\tilde{y}_1, \cdots, \tilde{y}_n) \quad \textit{where} \quad \tilde{y}_i = (1 - 2x_i)\,|y_i|,$$

where $\mathbf{y} = (y_1, \cdots, y_n) \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$ *and* $\mathbf{x} \leftarrow \mathsf{LDPC\text{-}PRC.Encode}(1^\lambda, \mathsf{pk})$.

- $\mathsf{Detect}_{\mathsf{sk}}(\mathsf{Img})$ *is an algorithm that outputs whether* Img *is watermarked.* Π_{GIM} *first extract a codeword* $\mathbf{x}'$ *from* Img *through Algorithm 7, and then run* $\mathsf{LDPC\text{-}PRC.Decode}(1^\lambda, \mathsf{sk}, \mathbf{x}')$. *If the result is* 1, Detect *outputs* true, *and* false *vise versa.*

We highlight that there are two main differences between Π_{GIM} and Π_{LLM}. First, Π_{GIM} initializes the key pair by generating a sparse vector $\mathbf{s}_i \in \mathbb{F}_2^{(n-r+i)}$ instead of $\mathbb{F}_2^{n-r}$ (Line 4 of Algorithm 1). Second, in order to embed extra messages and estimate the false positive rate, Π_{GIM} increases the column number of $\mathbf{G}$ from g to k by adding message bits and parity-check bits, which could be seen as a multi-bit PRC with its own implementation. Besides detecting the watermark, Π_{GIM} also introduces a $\mathsf{Decode}'()$ function to recover the original message $\mathbf{s}$ with the belief propagation algorithm, leveraging the sparsity of the secret key $\mathbf{P}$. However, we will demonstrate that such a modification can cause vulnerability, as shown in Section 7.3. The complete construction and implementation details of Π_{GIM} can be found in [16].

4 Attacks Against LDPC-PRC

In this section, we propose three attacks and provide analyses of their complexities.

4.1 Attack-I: Partial Secret Key Recovery

Given a public key $\mathbf{G} \in \mathbb{F}_2^{n \times g}$, Attack-I aims to recover a set of row vectors $\{\mathbf{v}^{(j)}\} \subseteq \mathbb{F}_2^n$ of the secret key $\mathbf{P}$. Using these vectors, an adversary can distinguish whether a set of target vectors $\{\mathbf{x}^{(i)}\}$ is PRC-encoded by computing the distribution of $\langle \mathbf{v}^{(j)}, \mathbf{x}^{(i)} + \mathbf{z} \rangle$. The attack procedure is shown in Figure 1 and summarized in Algorithm 8 of Appendix B.

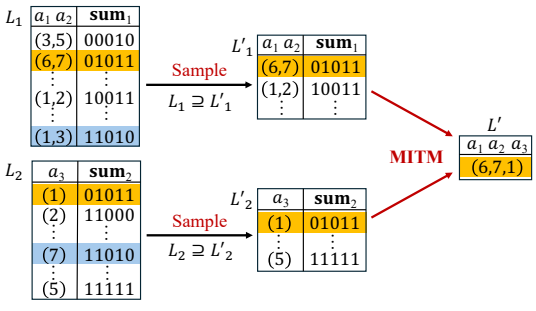


Figure 1: Illustration of the Meet-in-the-Middle (MITM) Attack utilized in Partial Secret Key Recovery ($t = 3$).

4.1.1 Pipeline of Attack-I

Stage-I: Low-Weight Kernel Vectors Recovery. Since Attack-I aims to compute a set of row vectors $\{\mathbf{v}^{(j)}\} \subseteq \mathbb{F}_2^n$ of the secret key $\mathbf{P}$ which satisfy $\mathbf{v}^{(j)}\mathbf{G} = \mathbf{0}$ and $w_H(\mathbf{v}^{(j)}) = t$, we adapt the Information Set Decoding (ISD) algorithm [5] to recover such a set of short vectors using a Meet-In-The-Middle (MITM) approach.

Let $n_1 = \lceil n/2 \rceil$, $n_2 = n - n_1$, $t_1 = \lceil t/2 \rceil$, and $t_2 = t - t_1$. For simplicity, we assume n is even, so $n_1 = n_2 = n/2$. Let L be the set of vectors in $\mathbb{F}_2^n$ satisfying $w_H(\mathbf{v}_{1:n_1}) = t_1$ and $w_H(\mathbf{v}_{n_1+1:n}) = t_2$ for all $\mathbf{v} \in L$. The probability that a random vector of weight t falls into L is $q = \binom{n_1}{t_1}\binom{n_2}{t_2}/\binom{n}{t}$. Thus, the expected number of rows of $\mathbf{P}$ in L is $r' = qr$, since each row of $\mathbf{P}$ is independently sampled at random.

To recover the expected r' rows, we construct two sets L_1 and L_2, where L_1 contains all $\binom{n}{t_1}$ sums of t_1 distinct rows from the upper half of $\mathbf{G}$ and L_2 contains all $\binom{n}{t_2}$ possible sums of t_2 distinct rows from the bottom half of $\mathbf{G}$: $L_1 = \{(a_1, ..., a_{t_1}, \mathbf{sum}_1) \mid 1 \le a_1 < \cdots < a_{t_1} \le n, \mathbf{sum}_1 = \sum_{i=1}^{t_1} \mathbf{G}_{a_i}\}$, $L_2 = \{(a_{t_1+1}, ..., a_t, \mathbf{sum}_2) \mid 1 \le a_{t_1+1} < \cdots < a_t \le n, \mathbf{sum}_2 = \sum_{i=t_1+1}^{t} \mathbf{G}_{a_i}\}$, where $\mathbf{G}_{a_i}$ denotes the a_i-th row of $\mathbf{G}$. We then employ the MERGE-JOIN algorithm [5] (a MITM implementation) to construct the target set

$$L = \{(a_1, ...a_{t_1}, a_{t_1+1}, ..., a_t) \mid \mathbf{sum}_1 + \mathbf{sum}_2 = \mathbf{0}\},$$

where $(a_1, ..., a_{t_1}, \mathbf{sum}_1) \in L_1$ and $(a_{t_1+1}, ..., a_t, \mathbf{sum}_2) \in L_2$. Thus, L stores the r' rows of the secret key $\mathbf{P}$.

To improve efficiency, we reduce L_1 and L_2 to smaller sets, ensuring the expected number of recovered rows from $\mathbf{P}$ is $l < r'$:

- When t is **even**, we randomly choose subsets $L'_i \subset L_i$ for $i = 1, 2$, with $|L'_1| = \lceil \frac{1}{\sqrt{r'/l}} \binom{n_1}{t/2} \rceil$ and $|L'_2| = \lceil \frac{1}{\sqrt{r'/l}} \binom{n_2}{t/2} \rceil$. Using MITM, we construct:

  $$L' = \{(a_1, ..., a_t) \mid \mathbf{sum_1} + \mathbf{sum_2} = \mathbf{0}\} \subseteq L,$$

  where $(a_1, ..., a_{t_1}, \mathbf{sum_1}) \in L'_1$ and $(a_{t_1+1}, ..., a_t, \mathbf{sum_2}) \in L'_2$. The expected number of rows of $\mathbf{P}$ in L' is $\frac{qr}{\sqrt{r'/l}\sqrt{r'/l}} = l$.

- When t is **odd**, we tailor L'_1 and L'_2 according to their sizes. Let $|L'_1| = \alpha\binom{n/2}{t_1}$ and $|L'_2| = \beta\binom{n/2}{t_2}$, where $\alpha = \sqrt{\frac{t_1}{(n/2-t_2)r'/l}}$ and $\beta = \frac{(n/2-t_2)\alpha}{t_1}$, ensuring $|L'_1| = |L'_2|$ and $\alpha\beta = l/r'$. Similarly, we construct L' via MITM, with the expected number of rows of $\mathbf{P}$ in L' equal to $\alpha\beta r' = l$.

Stage-II: Distinguishing Vectors. After recovering the rows $\{\mathbf{v}^{(1)}, \cdots, \mathbf{v}^{(l)}\}$ of the secret key $\mathbf{P}$, we now can determine whether vectors $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ are LDPC-PRC-encoded codewords or plain vectors. We assume these m vectors are sampled independently.

We first compute $\{\langle \mathbf{v}^{(j)}, \mathbf{x}^{(i)} + \mathbf{z}\rangle\}_{1\le j\le l, 1\le i\le m}$ and then calculate the ratio of zeros in the result set. We can choose a threshold τ and conclude that the m vectors are LDPC-PRC-encoded if the ratio exceeds τ, or plain vectors otherwise.

If the m vectors are LDPC-PRC-generated, i.e., $\mathbf{x}^{(i)} + \mathbf{z} = \mathbf{G}\mathbf{s} + \mathbf{e}^{(i)}$ with $\mathbf{e}^{(i)} \leftarrow \mathrm{Ber}(n, \omega)$, then

$$p := \Pr[\langle \mathbf{v}^{(j)}, \mathbf{x}^{(i)} + \mathbf{z}\rangle = 0] = \sum_{\tilde{j} \text{ is even}} \binom{t}{\tilde{j}} \cdot \omega^{\tilde{j}} \cdot (1-\omega)^{t-\tilde{j}}.$$

As proved in [8], there exists a constant $\delta > 0$ such that $p > 1/2 + \delta$. Then, we can calculate the expected True Positive Rate (TPR), i.e., the probability of correctly identifying m LDPC-PRC-encoded vectors, as:

$$\mathsf{TPR} = \sum_{j \ge \tau \cdot ml} \binom{ml}{j} \cdot p^j \cdot (1-p)^{ml-j},$$

and the False Positive Rate (FPR), i.e., the probability of incorrectly identifying m plain vectors as encoded, by:

$$\mathsf{FPR} = \sum_{j \ge \tau \cdot ml} \binom{ml}{j} \cdot \left(\frac{1}{2}\right)^{ml}.$$

To this point, we can choose m and τ to achieve target TPR and FPR values. For example, to achieve $\mathsf{TPR} = 1 - \mathsf{negl}(\lambda)$ and $\mathsf{FPR} = \mathsf{negl}(\lambda)$, we can set l, m such that $ml = r$ and $\tau = \frac{1}{2} + r^{-1/4}$, aligning with the Decode() function in the original Construction 1.

4.1.2 Complexity of Attack-I

We then analyze the time and data complexity of Attack-I based on the MITM partial recovery and the distinguishing procedure.

Theorem 1 (Complexity of Attack-I)**.** *Given a public key* $\mathbf{G}$ *and* $m = \Theta(n)$ *input vectors, there exists an adversary* $\mathcal{A}$ *that can distinguish against the undetectability property of* LDPC-PRC *with advantage* $|\Pr[\mathsf{Enc}] - \Pr[\mathsf{Uni}]| \ge 1 - \mathsf{negl}(\lambda)$ *in time complexity*

$$\mathrm{T}_{\mathsf{partial}} = O(1) \cdot g \cdot \alpha \cdot \binom{n/2}{\lceil t/2 \rceil} \tag{1}$$

and data complexity $\Theta(n)$*, where* $\alpha = 1/\sqrt{qr}$ *for even* t *and* $\alpha = \sqrt{\frac{\lceil t/2 \rceil}{qr(n/2 - \lfloor t/2 \rfloor)}}$ *for odd* t*, with* $q = \binom{n/2}{\lfloor t/2 \rfloor}\binom{n/2}{\lceil t/2 \rceil}/\binom{n}{t}$.

Proof. The data complexity is clearly $m = \Theta(n)$. For the time complexity, we can set $l = r/m = O(1)$ to achieve advantage $1 - \mathsf{negl}(\lambda)$ as discussed. Attack-I then takes time $\mathrm{T}_{\mathsf{partial}} + \Theta(n)$, where $\mathrm{T}_{\mathsf{partial}}$ is the time to recover rows of $\mathbf{P}$ and $\Theta(n)$ is the distinguishing complexity. Note that $\mathrm{T}_{\mathsf{partial}} = g \cdot \max\{|L'_1|, |L'_2|\}$, where the g factor comes from summing rows of $\mathbf{G}$. When t is **even**, $|L'_1| = |L'_2| = \frac{1}{\sqrt{r'}}\binom{n/2}{t/2}$, so $\mathrm{T}^{\mathsf{even}}_{\mathsf{partial}} = \frac{g}{\sqrt{r'}} \cdot \binom{n/2}{t/2}$. When t is **odd**, $|L'_1| = \alpha\binom{n/2}{t_1}$ and $|L'_2| = \beta\binom{n/2}{t_2}$. To minimize MITM complexity, we set $|L'_1| = |L'_2|$, i.e., $\alpha\binom{n/2}{t_1} = \beta\binom{n/2}{t_2}$, with $\alpha\beta = l/r' = O(1)/r'$. Thus, $\mathrm{T}^{\mathsf{odd}}_{\mathsf{partial}} = g \cdot \alpha \cdot \binom{n/2}{t_1}$, where $\alpha = \sqrt{\frac{\lceil t/2 \rceil}{qr(n/2 - \lfloor t/2 \rfloor)}}$. □

4.2 Attack-II: Weak Key Distinguisher

In the following, we present an efficient *multi-target* distinguishing attack that exploits an implementation vulnerability in the KeyGen() procedure rather than breaking the underlying LDPC-PRC primitive. In a multi-target attack, the adversary attempts to compromise any one vulnerable key from a set of multiple public keys. Specifically, the current PRC-based watermarking schemes (e.g., Π_{LLM} [8] and Π_{GIM} [16]) use Algorithm 1 or its variant to generate the random matrix $\mathbf{P} \xleftarrow{\$} \mathbb{F}_2^{r \times n}$, where each row of $\mathbf{P}$ has weight t. However, we find that rows of $\mathbf{G}$ can exhibit relationships that directly facilitate watermark distinguishing. This attack focuses on the simplest case where some rows of $\mathbf{G}$ are identical, and we call such $\mathbf{G}$ a weak key. The attack procedure is shown in Figure 2 and summarized in Algorithm 9.

4.2.1 Pipeline of Attack-II

Stage-I: Identifying Weak Keys. Weak keys can be identified by detecting whether $\mathbf{G}$ contains identical rows, which can be accomplished in $O(ng)$ time per key using hash tables. We now analyze the probability that a weak key is generated in Π_{LLM} [8] and Π_{GIM} [16].

Let $\mathrm{P}^{\mathrm{LLM}}_{\mathsf{weak}}$ denote the weak key probability for the LLM scheme Π_{LLM} [8]. Since $\mathbf{G}$ is generated using Algorithm 1, recall that it first samples a uniformly random matrix $\mathbf{G}_0$ with $n - r$ rows, then constructs the remaining r rows iteratively. Index these r rows from 1 to r. Ignoring the permutation Π, each of the last r rows of $\mathbf{G}$ is the sum of $t - 1$ rows from the first $n - r$ rows of $\mathbf{G}_0$. The number of possible distinct rows is $N = \binom{n-r}{t-1}$. Let p_i denote the probability that the first i rows are distinct. Then $p_1 = 1$ and $p_{i+1} = p_i \cdot \frac{N-i}{N}$. Thus, $p_r = \frac{N(N-1)\cdots(N-r+1)}{N^r}$, and

$$\mathrm{P}^{\mathrm{LLM}}_{\mathsf{weak}} = 1 - \frac{N(N-1)\cdots(N-r+1)}{N^r}. \tag{2}$$

For $\mathrm{P}^{\mathrm{GIM}}_{\mathsf{weak}}$, the key generation implementation differs from Algorithm 1 and has a larger enumeration space. The i-th row is a combination of the previous $i-1$ rows and the $n - r$ rows of $\mathbf{G}_0$. Let p'_i denote the probability that the first i rows are distinct. Then $p'_1 = 1$ and $p'_{i+1} = p'_i \cdot \left(1 - \frac{i}{\binom{n-r+i}{t-1}}\right)$. Therefore

$$\mathrm{P}^{\mathrm{GIM}}_{\mathsf{weak}} = 1 - \prod_{i=1}^{r} \left(1 - \frac{i-1}{\binom{n-r+i-1}{t-1}}\right). \tag{3}$$

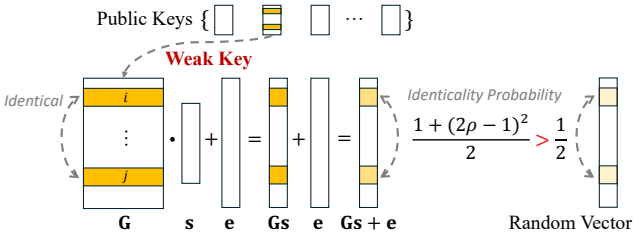


Figure 2: The workflow of Weak Key Distinguisher.

Stage-II: Distinguishing Vectors. Assume we have l pairs of row indices $\{(\alpha_j, \beta_j)\}$ such that $\mathbf{G}_{\alpha_j} = \mathbf{G}_{\beta_j}$ for all $1 \le j \le l$. For any $\mathbf{s} \in \mathbb{F}_2^{g\times 1}$, the α_j-th and β_j-th positions of $\mathbf{Gs}$ are always equal. This holds because $\mathbf{Gs}$ computes a linear combination of the columns of $\mathbf{G}$, and identical rows ensure identical positions in the result.

Although the output $\mathbf{Gs}+\mathbf{e}$ is perturbed by the noise vector $\mathbf{e}$, the adversary can construct a distinguisher with constant advantage. Let $\mathbf{e}_{\alpha_j} \leftarrow \mathrm{Ber}(\rho)$ and $\mathbf{e}_{\beta_j} \leftarrow \mathrm{Ber}(\rho)$ with $\rho < \frac{1}{2}$. The probability that positions α_j and β_j in $\mathbf{Gs}+\mathbf{e}$ are equal is $\rho^2 + (1-\rho)^2 = \frac{1+(2\rho-1)^2}{2}$. For a random vector, this probability is $\frac{1}{2}$. The statistical difference is $\frac{(2\rho-1)^2}{2}$, which is a constant number.

Using a threshold τ similar to Section 4.1, we can distinguish $\mathbf{Gs}+\mathbf{e}$ from a random vector in linear time, breaking the undetectability of LDPC-PRC. In practice, we can collect vectors $\{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$ from the oracle and calculate the ratio of vectors where positions α_j and β_j are equal. If this ratio exceeds τ, we conclude that the oracle uses LDPC-PRC encoding. When $ml = r$ and $\tau = \frac{1}{2} + r^{-1/4}$, the distinguishing advantage is overwhelmingly high due to a deduction same to that in Section 4.1.

4.2.2 Complexity of Attack-II

We then proceed to evaluate the time and data complexity of Attack-II based on the probability of encountering a weak key and the distinguishing procedure.

Theorem 2 (Time Complexity of Attack-II)**.** *There exists an adversary $\mathcal{A}$ such that, for a* fixed *key, runs in time $O(1)\cdot n\cdot g$. If the key is weak, $\mathcal{A}$ correctly identifies it and achieves a distinguishing advantage $|\Pr[\mathsf{Enc}] - \Pr[\mathsf{Uni}]| \ge 1 - \mathsf{negl}(\lambda)$ using $\Theta(n)$ input vectors.*

Since a weak key is encountered after P^{-1} independent keys in expectation, the total expected time complexity is

$$\mathrm{T}_{\mathsf{dis}} = O(1)\cdot \mathrm{P}^{-1}\cdot n\cdot g, \tag{4}$$

where $\mathrm{P} = \mathrm{P}_{\mathsf{weak}}^{\mathrm{LLM}}$ in Equation (2) for Π_{LLM} and $\mathrm{P} = \mathrm{P}_{\mathsf{weak}}^{\mathrm{GIM}}$ in Equation (3) for Π_{GIM}.

Proof. The expected cost of identifying a weak key is $O(1)\cdot \mathrm{P}^{-1}\cdot n\cdot g$, where $O(n\cdot g)$ is the cost of detecting duplicate rows in a single key, and the P^{-1} is the number of expected keys. Given a weak key $\mathbf{G}$, the adversary can distinguish $\mathbf{Gs}+\mathbf{e}$ from a random vector in linear time. A distinguishing advantage of $1-\mathsf{negl}(\lambda)$ is achieved by processing $\Theta(n)$ input vectors. Therefore, the total expected time complexity is $\mathrm{T}_{\mathsf{dis}} = O(1)\cdot \mathrm{P}^{-1}\cdot n\cdot g + \Theta(n)$. The $\Theta(n)$ term is omitted from the final complexity as it is dominated by the first term. □

4.3 Attack-III: Noise Overlay Attack

For PRCs, the noise vector is modeled as a Bernoulli noise channel as defined in Definition 1. This noise can originate from multiple sources, such as the inherent noise in LLM or GIM applications. For instance, in LLM watermarking, noise arises during the sampling of t_i from (p_i, x_i) by Algorithm 5, while in GIM, noise is introduced by the noise inversion procedure (i.e., $z(\cdot)$ of Algorithm 7). However, we highlight that an adversary may also actively modify content to evade detection. To investigate this threat, we propose a noise overlay attack that challenges the robustness of PRCs. The attack procedure is shown in Figure 3 and summarized in Algorithm 10.

4.3.1 Pipeline of Attack-III

Given an encoded vector $\mathbf{x} = \mathbf{Gs}+\mathbf{e}+\mathbf{z}$ and public key $\mathsf{pk} = (\mathbf{G}, \mathbf{z})$, the goal of Attack-III is to first recover the noise vector $\mathbf{e}$, then construct a malicious noise vector $\mathbf{e}'$ such that $\mathbf{x}+\mathbf{e}'$ cannot be successfully decoded. Since [8] does not specify an explicit noise rate bound, our objective is to maximize the resulting noise rate by adding $\mathbf{e}'$ with a *fixed* noise rate. We assume the decoding process fails once the final noise rate exceeds the designed threshold. Note that such a malicious noise attack is non-trivial because, under the same fixed noise rate, a PRC-based codeword with additional random noise of identical weight can be successfully decoded with high probability, while our malicious noise vector causes decoding to fail. We now detail the two steps of the attack.

- **Step I: Noise Recovery.** Upon receiving $\mathbf{x}$, we first compute $\mathbf{x}' = \mathbf{x}+\mathbf{z} = \mathbf{Gs}+\mathbf{e}$. Let $\mathbf{H} \in \mathbb{F}_2^{(n-g)\times n}$ be the dual parity-check matrix of $\mathbf{G} \in \mathbb{F}_2^{n\times g}$ satisfying $\mathbf{HG} = \mathbf{0}$.

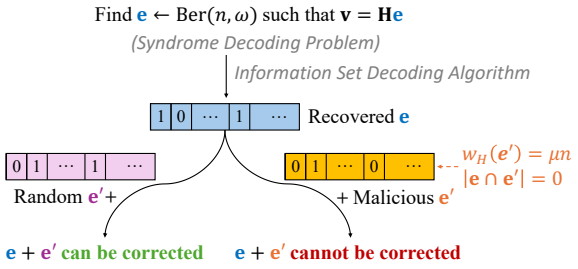


Figure 3: The workflow of Noise Overlay Attack.

Next, we can get $\mathbf{v} = \mathbf{H}\mathbf{x}' = \mathbf{H}\mathbf{G}\mathbf{s} + \mathbf{H}\mathbf{e} = \mathbf{H}\mathbf{e}$. At this stage, we need to find a vector $\mathbf{e} \leftarrow \mathrm{Ber}(n, \omega)$ satisfying $\mathbf{H}\mathbf{e} = \mathbf{v}$. This reduces to the classical Syndrome Decoding (SD) problem [6], which is computationally equivalent with the LPN problem in LDPC-PRC. Consequently, it can be solved using ISD algorithms [5, 11, 29, 30, 36, 42]. Although the exact weight of $\mathbf{e}$ is unknown, we consider the upper bound of error-correcting capability where the noise rate equals $\frac{1}{2} - \varepsilon$, as calculated by the revised inequality in Lemma 1.

- **Step II: Noise Overlay.** Recall that the received vector is $\mathbf{x}' = \mathbf{G}\mathbf{s} + \mathbf{e}$ where $\mathbf{e} \leftarrow \mathrm{Ber}(n, \omega)$. If we add another random $\mathbf{e}' \leftarrow \mathrm{Ber}(n, \mu)$ to $\mathbf{x}'$, where $\mathbf{e}$ and $\mathbf{e}'$ are independent, then $\Pr[e_i + e'_i = 1] = \omega + \mu - 2\omega\mu$ and $\mathbb{E}[w_H(\mathbf{e} + \mathbf{e}')] = (\omega + \mu - 2\omega\mu)n$. However, once $\mathbf{e}$ is recovered, we can maliciously construct a special $\mathbf{e}'$ satisfying both $w_H(\mathbf{e}') = \mu n$ and $|\mathbf{e} \cap \mathbf{e}'| = 0$, such that

$$\mathbb{E}[w_H(\mathbf{e} + \mathbf{e}')] = (\omega + \mu)n \gg (\omega + \mu - 2\omega\mu)n.$$

  Assume LDPC-PRC is designed to tolerate additional random noise with rate μ_0, which corresponds to an overall noise rate of $(\omega + \mu_0 - 2\omega\mu_0)$. Hence, we can set $\mu = \mu_0$ in our attack. While the resulting randomly combined noise remains detectable, the adversarially constructed noise vector $\mathbf{e} + \mathbf{e}'$ has weight $(\omega + \mu_0)n$, exceeding the decoding capability of the LDPC-PRC. Therefore, introducing $\mathbf{e}'$ successfully causes decoding failures.

4.3.2 Complexity of Attack-III

In this part, we derive the time and data complexity of Attack-III in Theorem 3. In particular, we adopt the Prange algorithm [36] to estimate the time complexity.

Theorem 3 (Complexity of Attack-III)**.** *Given a public key* pk *and an encoded vector* $\mathbf{x}$*, there exists an adversary that can construct* $\mathbf{x}'' = \mathcal{E}(\mathbf{x})$ *to achieve* $\mathsf{Decode}(1^\lambda, \mathsf{sk}, \mathbf{x}'') \neq \mathsf{m}$ *with time complexity*

$$\mathrm{T}_{\mathsf{overlay}} = \left(\frac{1}{2} + \varepsilon\right)^{-g} \cdot n^3. \tag{5}$$

Proof. For an SD problem $\mathbf{H}\mathbf{e} = \mathbf{v}$, we first randomly shuffle the columns of $\mathbf{H}$. We then calculate the probability that this shuffle concentrates all nonzero elements of $\mathbf{e}$ in the first $n - g$ positions, leaving the last g positions as zeros (i.e., $w_H(\mathbf{e}_{n-g+1:n}) = 0$). Since each element of $\mathbf{e}$ equals 1 with probability $\frac{1}{2} - \varepsilon$, this probability is

$$\mathrm{P}_{\mathsf{nz}} = \left(\frac{1}{2} + \varepsilon\right)^{g}.$$

If this condition holds, the SD problem can be solved via Gaussian elimination with n^3 operations. Thus, the total time complexity for the Prange algorithm is $\mathrm{T}_{\mathsf{Prange}} = \mathrm{P}_{\mathsf{nz}}^{-1} n^3$. Since the remaining steps of adding $\mathbf{e}'$ require only $O(n) \ll \mathrm{T}_{\mathsf{Prange}}$ time, the total complexity is dominated by noise recovery

$$\mathrm{T}_{\mathsf{overlay}} = \left(\frac{1}{2} + \varepsilon\right)^{-g} n^3.$$

□

Table 2: Parameters of LDPC-PRC for Π_{LLM} and Π_{GIM}.

	n	r	g	λ	t
Π_{LLM} [8]	-	$0.99n$	$c\log^2 n$	–	$\log n$
Π_{GIM} [16]	2^{14}	$n-k-\lambda$	$\log\binom{n}{t}$	g	$3 \le t \le 7$
Config 1 (for Π_{LLM})	2^{17}	$0.99n$	$\log\binom{n}{t}$	g	$3 \le t \le 14$
Config 2 (for Π_{GIM})	2^{14}	$n-k-\lambda$	$\log\binom{n}{t}$	g	$3 \le t \le 7$

Remark. In Π_{GIM}, we note that the weight of added noise η is much lower than the theoretical bound $\frac{1}{2} - \varepsilon$. Let $\mathrm{T}'_{\mathsf{overlay}}$ denote the cost of Attack-III with the concrete noise rate η instead of $\frac{1}{2} - \varepsilon$. Consequently, we can estimate $\mathrm{T}'_{\mathsf{overlay}}$ as

$$\mathrm{T}'_{\mathsf{overlay}} = (1 - \eta)^{-g} \cdot n^3.$$

5 Concrete Time Complexity Analysis

Building on our theoretical analysis, we now analyze the complexity of our attacks under concrete parameter choices of watermarking schemes. Although the attacks initially operate on watermarked images or texts, we first transform them into the corresponding codewords via Algorithm 6 and Algorithm 7. As the transformation cost is negligible compared to the attacks themselves, we focus solely on the complexity of our three attacks.

5.1 Parameter Configurations

We summarize the parameter configurations of LDPC-PRC for Π_{LLM} and Π_{GIM} in Table 2, where the first two rows present the default settings provided by the original Π_{LLM} and Π_{GIM}. However, we notice that some critical parameter settings remain unspecified, e.g., n and λ in Π_{LLM}. Therefore, we determine two parameter configurations, Config 1 and Config 2, for Π_{LLM} and Π_{GIM}, respectively. These specific parameters facilitate a concrete time complexity analysis of the proposed attacks.

5.1.1 Parameter Configurations for Π_{LLM}

Config 1. *For* Π_{LLM}*, we set* $n \le 2^{17}$*,* $r \le 0.99n$*,* $g = \lambda = \log\binom{n}{t}$*, and* $3 \le t \le 14$*.*

When analyzing the computational complexity of attacks against Π_{LLM}, we use the maximum parameter values in Config 1 to provide the upper bound of security: $n = 2^{17}, r = \lfloor 0.99n \rfloor$, and $g = \lambda = \log \binom{n}{t}$ for each t. Next, we explain the rationales for the parameter settings in Config 1 as follows:

- **The code length** n: According to Construction 2, each token is mapped into binary bit strings. We assume the length of binary bit strings is smaller than 32 bits, i.e., the size of the token alphabet[1] is smaller than 2^{32}. Meanwhile, we assume that the max output length is $2^{12} = 4,096$ tokens, which is a common setting for LLMs (e.g., gpt-3.5-turbo [31] and gpt-4-turbo [32]).[2] Note that in practical scenarios, the output length of LLMs typically does not reach the maximum limit. If the defender requires stronger robustness against the binary deletion channel (BDC) as described in [8], the LDPC-PRC must be embedded into a longer code. Consequently, the length of LDPC-PRC becomes significantly shorter than the output length of LLMs. As a result, it is natural and reasonable to assume that the code length n is not larger than $32 \times 2^{12} = 2^{17}$.

- **The column size** g **and security parameter** λ: Though [8] sets $g = c\log^2 n$ where c is a constant, but no concrete value of c and expression of λ is given. Hence, we follow Π_{GIM} [16] to set $g = \lambda = \log \binom{n}{t}$.

- **The Row-wise nonzero count** t: We set $3 \leq t \leq 14$ for Π_{LLM} due to the following reasons. (i) The lower bound of t: When t is 1 or 2, the complexity of Attack-I is linear in n. Therefore, to exclude this trivial attack, we restrict our analysis to $t \geq 3$. (ii) The upper bound of t: We use Lemma 1 to calculate the upper bound of t. Let $\rho = 1/2 - \varepsilon$ be the maximum error-correcting capability. Since no explicit value of ρ is discussed in [8], we use the experimental results in Π_{GIM} as reference. During the experiments in Section 6.3, we find that Π_{GIM} has to correct vectors with noise rate $\rho = 0.074$. Hence, we bound $\rho \geq 0.074$, i.e., $\varepsilon \leq 0.426$. However, when we try to calculate the upper bound of t with r and ε following [8], we notice a mistake in their calculation, i.e.,

  $$(2\varepsilon)^t/2 > r^{-1/4} \nLeftrightarrow t < 1 + \log(r)/4\log(1/2\varepsilon).$$

  Therefore, we give a corrected version in Lemma 1 and get $t \leq 14$.

[1] The size of the alphabet varies from thousands to millions; thus, 2^{32} is a very loose upper bound.

[2] We also notice that the recently released, more advanced LLMs are capable of supporting 32,768 max output tokens, e.g., gpt-4.1-turbo [33]. We will prove that Π_{LLM} can still not reach an 128-bit security even with such length. More details are available in Section 7.1.

Lemma 1. *In order to decode successfully with high probability, we need* $(2\varepsilon)^t/2 > r^{-1/4}$*, i.e.,*

$$t < \frac{\frac{1}{4}\log r - 1}{\log(1/2\varepsilon)}.$$

Proof. With $(2\varepsilon)^t/2 > r^{-1/4}$, we have $(2\varepsilon)^t > 2r^{-1/4}$. Since $0 < \varepsilon < 1/2$, we can simplify the inequality to $t < \log_{2\varepsilon}(2r^{-1/4})$, i.e., $t < \frac{1-1/4\log r}{\log(2\varepsilon)} = \frac{\frac{1}{4}\log r - 1}{\log(1/2\varepsilon)}$. □

5.1.2 Parameter Configurations for Π_{GIM}

Config 2. *For* Π_{GIM}*, we set* $n = 2^{14}$*,* $g = \lambda = \log \binom{n}{t}$*,* $k = \lambda + message_bit + parity_bit$ *where* $message_bit = 512$*, and* $parity_bit = \log_2(\text{FPR})$ *where* $\text{FPR} = 10^{-5}$*. When calculating* $\mathbf{y} = \mathbf{Gs} + \mathbf{e}$*, the added noise vector* $\mathbf{e}$ *has noise weight* $\eta = 1 - 2^{-\lambda/g^2}$.

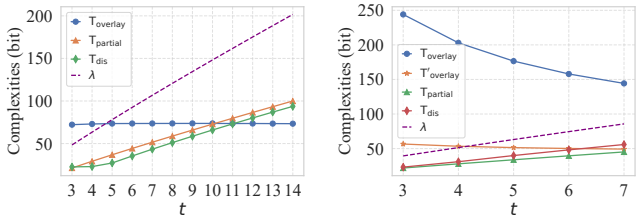
For Config 2, we employ the default parameters from [16]. Note that the weight t is set to 3 in most cases discussed in [16], and it is also suggested to set $t = \log(n)/2 = 7$ to ensure cryptographic undetectability. For a complete analysis of our attacks, we vary the weight t from 3 to 7 and change r, g, k, λ, and η accordingly.

5.2 Overall Performance

First of all, we present a comprehensive comparison between the complexity of our attacks and the claimed security levels of LDPC-PRC, and illustrate the trend of complexity with varying parameters in Figure 4. The detailed complexities are available in Table 6 and Table 7 (Appendix C).

Performance on Π_{LLM}**.** First, as shown in Figure 4a, the best complexities of the three attacks are lower than the claimed security level λ across all parameters. Second, it presents clear trends in the complexities. For Attack-I and Attack-II, their respective attack complexities (i.e., T_{partial}, and T_{dis}) exhibit a consistent increasing trend with growing t. Nevertheless, it consistently remains below 128 bits, demonstrating that our attacks remain feasibly threatening. Notably, as shown in Table 6, for Attack-II, the frequency of weak keys is non-negligible and approaches 100% for parameters $t = 3$ and $t = 4$, making the attack always effective in these cases. For Attack-III, we can observe that even as t increases, the value of T_{overlay} remains nearly unchanged and consistently stays around 73 bits, indicating that Π_{LLM} exhibits vulnerability to our noise overlay attack across a reasonable range of parameters.

Performance on Π_{GIM}**.** Similarly, Figure 4b shows that the best complexities of our three attacks remain below the claimed security level λ for all parameter settings. The first two attacks exhibit increasing complexity with growing t, which is similar to the performance on Π_{LLM}. Although $P_{\text{weak}}^{\text{GIM}}$ of GIMs is larger than that of LLMs, it remains high under



(a) LLM. (b) GIM.

Figure 4: Time complexities of all attacks. The complexities are expressed in logarithm form.

concrete parameters. In Table 7, when $t = 3$, weak keys occur with nearly 100%, indicating practical vulnerability. For Attack-III, we can observe that both the complexity $\mathrm{T}_{\mathsf{overlay}}$ and $\mathrm{T}'_{\mathsf{overlay}}$ decrease as t increases, which comes from the decrease in the theoretical noise rate ρ and the concrete noise rate η. Due to the gap between ρ and η , there is a huge difference between $\mathrm{T}_{\mathsf{overlay}}$ and $\mathrm{T}'_{\mathsf{overlay}}$. Taking into account $\mathrm{T}'_{\mathsf{overlay}}$ instead of $\mathrm{T}_{\mathsf{overlay}}$, all complexities are below 64 bits, indicating practical threats under the parameter choices of Π_{GIM}.

5.3 Further Analysis of Attack-III

Performance on Π_{LLM}. The experimental results show that $\mathrm{T}_{\mathsf{overlay}}$ increases slightly as t increases when $3 \leq t \leq 10$, but decreases when $t \geq 12$. This phenomenon indicates that a larger t does not always lead to a higher security level. We also estimate all reasonable parameters on $t > 14$, which concretely verifies this conclusion. We further analyze this phenomenon on the basis of the following truth.

- $\mathrm{T}_{\mathsf{overlay}} \propto (1-\rho)^{-g} = (\frac{1}{2}+\varepsilon)^{-g}$.
- g increases as t increases since $g = \log \binom{n}{t}$.
- ε increases as t increases because of the constraint in Lemma 1.

Therefore, for small values of t, the term g dominates the behavior of $\mathrm{T}_{\mathsf{overlay}}$, causing $\mathrm{T}_{\mathsf{overlay}}$ to increase as t increases. As t progressively increases, ε gradually becomes dominant, leading to a decrease in $\mathrm{T}_{\mathsf{overlay}}$. Second, although the estimated $\mathrm{T}_{\mathsf{overlay}}$ seems large for small t compared to the other attacks, the defender might add a much lower noise rate as Π_{LLM} does, thereby causing a much lower $\mathrm{T}_{\mathsf{overlay}}$.

Performance on Π_{GIM}. $\mathrm{T}_{\mathsf{overlay}}$ decreases sharply when $3 \leq t \leq 7$ since the noise weight dominates the complexity. However, the actual noise weight $\eta \ll \rho$ in all cases, making $\mathrm{T}'_{\mathsf{overlay}}$ much lower than the theoretical bound $\mathrm{T}_{\mathsf{overlay}}$. The gap could be as large as 187 bits, indicating the practical threat of our attack.

6 Real-World Evaluations

To further demonstrate the practical threat posed by our attacks, we launch Attack-I and Attack-II against real-world large generative models. For Attack-III, the text quality of the watermarked Π_{LLM} is low, making it difficult to analyze the effect of our modification. Therefore, we conduct Attack-III only on Π_{GIM}. Moreover, to verify that our attacks can generalize beyond specific experimental settings, we also conduct supplementary experiments on different models in Appendix E.

6.1 Setups

Target LLM. We employ DeepSeek-R1-Distill-Qwen-7B [17] as our real-world target LLM. Each response is generated with a fixed length of $1,024$ tokens, chosen from a vocabulary of size 152,064. To embed watermarks, we apply Π_{LLM} [8] as described in Section 3.3. Following Config 1, we set $n = \lceil \log_2(152,064) \rceil * 1,024 = 18,432$, $r = \lfloor 0.95n \rfloor = 17,510$, and t as 3 or 4.

Target GIM. We adopt stable-diffusion-2-1-base [38] as our target GIM. To generate images, we use the Multistep DPMSolver Scheduler [26] as the de-noising scheduler $f(\cdot)$ of Algorithm 4 and set the de-noising step s to 50. We apply the Π_{GIM} scheme to embed watermarks with $t = 3$ or 4, and all other parameters are set as in Config 2. During watermark verification, we perform DDIM Inversion [41] as the noise inversion scheduler $z(\cdot)$ of Algorithm 7 with the same number of steps and guidance scale as the image generation process.

AI-Generated Contents. In each set of parameters, we randomly generate 128 pairs of $\mathsf{pk} = (\mathbf{G}, \mathbf{z})$. For each pk, we generate 16 texts or images. Meanwhile, we also generate non-watermarked content with the same number as a control group. For GIMs, we follow the model hyperparameter settings in [16] and sample the prompts from the test split of the Gustavosta/Stable-Diffusion-Prompts dataset [39].

Settings of Attack-I and Attack-II. When attacking Π_{LLM}, we set the temperature to 1.8, and the parameters for attacking are set as follows: for $t = 3$, we set $|L'_1| = |L'_2| = 18,432$ and $\tau = 0.60$ for both Attack-I and Attack-II. When $t = 4$, we set $|L'_1| = |L'_2| = 5 \times 10^6$, and $\tau = 0.55$ for Attack-I and $\tau = 0.65$ for Attack-II. The parameter settings for attacking Π_{GIM} are as follows: When $t = 3$, we set $|L'_1| = |L'_2| = 16,384$ and $\tau = 0.75$ for both Attack-I and Attack-II. When $t = 4$, we set $|L'_1| = |L'_2| = 5 \times 10^5$ and $\tau = 0.75$ for both Attack-I and Attack-II.

Evaluation Metrics. (1) For Attack-I, we define the success rate as the probability that $|L'| > 0$. For Attack-II, the success rate corresponds to the probability of encountering weak keys. (2) We also evaluate the TPR and FPR for watermark detection upon the successful attacks. Let TPR_0 denote the TPR when testing the original codeword generated by the

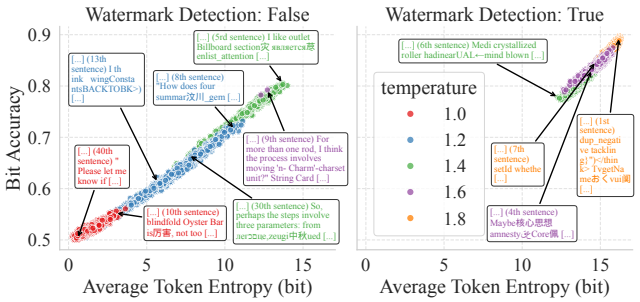


Figure 5: The relationship between the readability of generated text and the detectability of watermarks. We set $t = 3$ for Π_{LLM}.

defender, i.e., $\mathbf{x} = \mathbf{Gs} + \mathbf{e} + \mathbf{z}$, and FPR_0 denote the FPR when testing a uniformly random vector. Meanwhile, we use TPR_1 to denote the TPR for the extracted codeword from the watermarked content, which includes both the defender's added noise and the noise introduced by the schemes. Specifically, the additional noise of Π_{LLM} is introduced in Line 5 and Line 7 of Algorithm 5, while the noise of Π_{GIM} arises from the error of DDIM inversion. Correspondingly, FPR_1 represents the FPR for the codeword corresponding to the unwatermarked content.

6.2 Performance on LLMs

6.2.1 Impracticality of Π_{LLM}

Prior to an in-depth examination of the attack effectiveness, it is imperative to first elucidate the impracticality of Π_{LLM}. Recall in Algorithm 5, the tokens are biased according to the probability p' derived from the codewords, so the bit accuracy of the recovered message depends on the average token entropy of the LLM output. To satisfy LDPC-PRC 's correction bounds, the LLM must produce high-entropy token distributions, which correspond to low-confidence predictions and thus low-quality responses.

To examine this, we generate watermarked texts using temperature values temp from 1.0 to 1.8, controlling output entropy. As shown in Figure 5, the watermark is undetectable at temperatures of 1.0 and 1.2. At $\text{temp} = 1.4$, watermarks are detectable in 60% of outputs, but readability is severely degraded, with frequent language mixing (green texts in Figure 5). At $\text{temp} = 1.6$ and 1.8, detection rates further increase, but the generated text becomes nearly unreadable (purple and orange texts in Figure 5). Therefore, we launch attacks against texts capable of carrying watermarks, disregarding their readability.

6.2.2 Performance of Attack-I

As described in Table 3, Attack-I maintains a perfect success rate of 100%, which means we can stably construct nonempty

Table 3: Watermark detection attacks against real-world LLM.

Attack	t	TPR_0	FPR_0	TPR_1	FPR_1	Success Rate
I	3	100%	0%	61%	0%	100%
	4	100%	0%	98%	0%	100%
II	3	100%	0%	100%	0%	100%
	4	98%	10%	59%	10%	63%

L'. More precisely, we can recover an average of 8.8 rows of the secret keys. Meanwhile, the attainment of 100% TPR_0 alongside 0% FPR_0 confirms that Attack-I possesses a requisite precision to discriminate between PRC codewords and random vectors. However, the TPR_1 on the real-world output texts becomes 61% when $t = 3$ and 98% when $t = 4$. This degradation is attributable to the additional noise introduced during the token sampling procedure in Algorithm 6. Specifically, the error rate increases from 0.10 for the original codeword to 0.20 for the recovered codeword.

6.2.3 Performance of Attack-II

On the other hand, Attack-II achieves complete success at $t = 3$, yet drops to 63% when $t = 4$. It demonstrates a considerable likelihood of weak keys occurring within LLM scenarios. Furthermore, the high TPR_0 values and low FPR_0 values collectively demonstrate Attack-II's efficacy in differentiating between PRC codewords and random vectors, demonstrating that with up to 22.09 duplicated rows on average, the distinguisher performs very well. Unlike Attack-I, the TPR_1 sustains a perfect 100% on real-world texts at $t = 3$. When $t = 4$, the gap between TPR_1 and FPR_1 remains sufficiently large to enable reliable detection.

6.3 Performance on GIMs

6.3.1 Attack Scenario

Given an image, both defenders verifying watermarks and adversaries launching detection attacks must utilize Algorithm 7 to recover the codeword. Algorithm 7 employs two neural networks, `U-net` and `E-net`. The defender typically employs the same `U-net` utilized in the generation process, along with an `E-net` compatible with the `D-net` that was used for generating the image, for a highly precise codeword extraction. For the adversary, we first assume that they can obtain the same `U-net` and `E-net` as the defender. However, considering that model parameters are typically kept confidential as commercial secrets, we also consider that the adversary will employ a proxy model that differs from the defender's.

Table 4: Watermark detection attacks against real-world GIM.

Attack	t	TPR_0	FPR_0	TPR_1	FPR_1	Success Rate
I	3	99%	0%	93%	0%	100%
	4	93%	4%	66%	4%	78.9%
II	3	100%	0%	96%	0%	100%
	4	100%	0%	100%	0%	2.3%

6.3.2 Performance of Attacks

Distinguishing Attacks. The performance of distinguishing attacks against Π_{GIM} is shown in Table 4. In our experiments, we find the error rate of the original codeword is 0.018 and the noise rate of the received codeword is 0.074. With a lower-weight noise vector compared to Π_{LLM}, we achieve more effective attacks. When $t = 3$, although only 5.8 rows are recovered on average (vs. 8.8 in Π_{LLM}) in Attack-I, the TPR_1 is 93% (vs. 61% in Π_{LLM}) with FPR remaining 0%. For Attack-II, the weak key frequency is 100% and duplicated rows average 14.86. Our distinguisher also performs well with $\text{TPR}_1 = 96\%$ and $\text{FPR}_1 = 0\%$. The cases of $t = 4$ are similar to $t = 3$. Meanwhile, to verify that our attacks also work without access to the target model's neural network in real-world scenarios, we conduct extra experiments with other proxy models, and the results show that our attacks still succeed. Detailed results under different proxy models are discussed in Appendix D.

Removal Attacks. Then, we investigate leveraging Attack-III to remove the watermark embedded by Π_{GIM}. Given Attack-III, the attacker can recover $\mathbf{x} - \mathbf{e}$. Subsequently, Projected Gradient Descent (PGD) [28] is applied to search for an adversarial perturbation Err within δ-bounded l_∞ norm, such that the sign of the reversed codeword aligns with $-(\mathbf{x} - \mathbf{e})$. Such an optimization objective is formalized as

$$\arg\min_{\mathsf{Err}} \left\| \tanh[k \cdot \texttt{ImageToCodeword}(\mathsf{Img} + \mathsf{Err})] + (\mathbf{x} - \mathbf{e}) \right\|_2,$$

where k is the hyperparameter setting to 5. We conduct removal attacks on 100 images watermarked with Π_{GIM} under $t = 3$. After 100 optimization steps with a learning rate of 0.01, the watermark removal success rate is 67% under $\delta = 8/255$, 89% under $\delta = 12/255$, and 96% under $\delta = 16/255$, with the visual examples shown in Figure 6. These results demonstrate that Attack-III can be effectively leveraged to remove watermarks against Π_{GIM}.

7 Mitigations

7.1 Parameter Suggestions

We show how Attack-I and Attack-III can be mitigated through appropriate parameter selection in Theorem 4.



Figure 6: Example images with removed watermark.

Theorem 4 (Complexities under Different Parameters)**.** *Following the parameter relationships specified in Config 1 for Π_{LLM} and Config 2 for Π_{GIM}, while varying n and t, we obtain the following results by substituting the corresponding parameters into Theorem 1 and Theorem 3:*

- *For $(t, n) \in \{(9, 2^{32}), (11, 2^{26}), (13, 2^{23}), (15, 2^{20})\}$, we have $\text{T}_{\text{partial}} \leq 2^{128}$.*
- *When $n = 2^{24}$, we have $\text{T}_{\text{overlay}} \leq 2^{122.21}$ for all choices of t.*

For Attack-I, it is natural to increase n and t to achieve a higher security level. In Theorem 4, we provide the minimum code length n targeting at a security level of $\lambda = 128$ bits under different choices of t. However, it is noteworthy that increasing t also reduces the error-correcting capability of the LDPC-PRC. Therefore, a careful trade-off between security and robustness is required to effectively mitigate Attack-I.

For Attack-III, the most effective defense is to increase n rather than t. In Theorem 4, a code length $n > 2^{24}$ is necessary to achieve $\lambda \geq 128$. However, current state-of-the-art LLMs, such as GPT-4.1-Turbo [33], have a maximum token length of $2^{15} = 32{,}768$, which corresponds to $n = 2^{20}$. The situation is similar for Π_{GIM}. As discussed in [38], the maximum initial latent dimension is $4 \times 128 \times 128$, corresponding to a code length of only $n = 2^{16}$, still a significant gap from the target of 128-bit security. Consequently, constructing LDPC-PRC-based watermarking schemes for LLMs and GIMs that achieve 128-bit security remains a significant practical challenge.

If these parameter requirements are satisfied, the resulting security can be reduced to the hardness of the underlying LPN and planted XOR problems, thereby providing protection not only against our attacks but also against other potential attacks targeting these primitives.

7.2 Revision of Key Generation Algorithm

The key strategy for mitigating Attack-II is to modify the $\mathsf{KeyGen}()$ function to reduce the probability of generating weak keys. We present a revised $\mathsf{KeyGen}()$ algorithm in Algorithm 2, with modifications highlighted in **green**. The revised algorithm first samples the secret key randomly and then constructs the public key $\mathbf{G}$ via Gaussian elimination. This process preserves the required randomness assumptions

Algorithm 2 Revised Key Generation Algorithm

Input: Parameters: n, $r = 0.99n$, g, t
Output: Matrices $\mathbf{P}$ and $\mathbf{G}$ for watermark generation

1: Initialize an empty list for rows of $\mathbf{P}$
2: **for all** $i \in [0.99n]$ **do**
3: Sample a random t-sparse vector $\mathbf{s}_i \in \mathbb{F}_2^n$
4: Append $\mathbf{s}_i$ to the list of rows for $\mathbf{P}$
5: **end for**
6: Let $\mathbf{P}$ be the matrix whose rows are $\mathbf{s}_1, \ldots, \mathbf{s}_{0.99n}$
7: Perform Gaussian elimination on P and transform it to system form $[\mathbf{I}_r || \mathbf{P}_1]$
8: Let $\mathbf{G}_0 = [\mathbf{P}_1^T || \mathbf{I}_{n-r}]^T$
9: Random select g columns from $\mathbf{G}_0$ to construct $\mathbf{G} \in \mathbb{F}_2^{n \times g}$
10: Sample a random permutation $\Pi \in \mathbb{F}_2^{n \times n}$, and let $\mathbf{P} \leftarrow \mathbf{P}\Pi^{-1}, \mathbf{G} \leftarrow \Pi\mathbf{G}$
11: **return** $(\mathbf{P}, \mathbf{G})$

for both $\mathbf{P}$ and $\mathbf{G}$ in the LDPC-PRC framework, making the probability of generating weak keys negligible.

It should be noted that this mitigation introduces a computational trade-off, i.e., the key generation complexity increases from $O(n^2)$ in the original algorithm to $O(n^3)$, owing to the Gaussian elimination step in Line 7.

7.3 Implementation Suggestions for GIMs

As discussed in [8], the condition $t = \Omega(\log n)$ is necessary to guarantee the security of PRC-based constructions, as it ensures that the entropy of the secret key space remains sufficiently large to resist exhaustive search attacks. However, Π_{GIM} [16] introduces additional parity bits and employs a belief propagation decoding algorithm to recover the embedded message, which in turn enforces the parameter t to be a constant in order to achieve reliable decoding. When t is reduced from $\Omega(\log n)$ to a constant c, the row-wise entropy of the secret key space $\binom{r}{t}$ significantly degrades, shrinking to $\Omega(r^c) = \Omega(n^c)$. As a consequence, an adversary can mount a brute-force attack by enumerating all possible rows of the secret key with polynomial-time complexity $O(n^c)$. Therefore, we recommend following the design principle in [8] when constructing the multi-bit PRC and avoiding explicit decoding procedures such as $\mathsf{Decode}'()$.

Moreover, as discussed in Section 5.3, Π_{GIM} adds a noise vector with weight $\eta \ll \rho$, which results in an extremely efficient concrete attack compared to the theoretical complexity $\mathrm{T}_{\mathsf{overlay}}$. In fact, since $\lambda = g = \Theta(\log^2 n)$ and k could be seen as the sum of g and a constant c in Π_{GIM}, the complexity of Attack-III, $\mathrm{T}'_{\mathsf{overlay}} = 2^{k/g} n^3 = 2^{1+c/\Theta(\log^2 n)} n^3 = O(n^3)$, is only *polynomial*. Therefore, we suggest increasing the weight of the added noise vector.

8 Conclusion & Future Work

This paper presents the first cryptanalysis of PRC by proposing three attacks, including two distinguishing attacks and one noise overlay attack. Through rigorous theoretical analysis of success rates and complexity, we demonstrate that the current instantiation LDPC-PRC exhibits security vulnerabilities. Our attack complexity is lower than its claimed security guarantees, with most attacks requiring fewer than 128 bits of complexity. Furthermore, we implement PRC-based watermarking schemes in real-world large generative models to validate the practical threat. We also propose three defenses, including parameter and implementation recommendations, along with a revised key generation algorithm. Beyond security analysis, we refine PRC's parameter ranges and prove that even without attacks, the current PRC scheme faces challenges in practical LLM applications. Our findings will facilitate the development of future AIGC watermarking schemes that achieve robustness, undetectability, and practicality.

Future Work. Several open problems remain for future work. First, as discussed in Section 6.2, Π_{LLM} is not directly applicable to real-world scenarios. Since our work focuses on security analysis, we have not yet provided concrete measures to enable the practical deployment of PRC in large-scale LLMs. Second, the theoretical asymptotic analysis of PRC fails to provide security under practical feasible parameters. It might be promising to construct Π_{GIM} focusing only on robustness instead of undetectability. Finally, regarding parameters (n, g, t, r), because r is typically set slightly below n with negligible impact on security or error correction, we focus on n and t in our analysis of practical schemes. While we show that increasing g to k in Π_{GIM} yields no asymptotic advantage, the precise discussion of g remains open for further investigation.

Acknowledgement

We sincerely thank the anonymous reviewers for their constructive suggestions. This work was supported by the National Cyber Security-National Science and Technology Major Project (2026ZD1500900), the Scientific Research Innovation Capability Support Project for Young Faculty (ZYGXQNJSKYCXNLZCXM-P4), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM114), the National Natural Science Foundation of China (62402273), and Zhongguancun Laboratory.

Ethical Considerations

Our ethical considerations consist of *stakeholder analysis*, *impact analysis*, *mitigations*, and *justifications*.

Stakeholder Analysis. Our cryptanalysis of PRCs involves three primary stakeholder groups as follows. (1) *Model*

Providers: They provide large models, including LLMs and GIMs, to users. Upon receiving requests from users, these models generate watermarked text or images, where the watermark is only detectable by model providers holding the secret key. (2) *End Users*: They utilize large models to generate content that contains watermarks which embed user-related information, such as user identifiers and timestamps, which may raise privacy concerns. (3) *Researchers*: This group includes researchers from both the security and AI communities. They study whether watermarking mechanisms affect the quality of AI-generated content and whether the embedded watermarks may leak sensitive user information.

Impact Analysis. Our analysis has both positive and negative impacts for the three stakeholder groups identified.

Positive Impacts: (1) *Enhanced Deployment Security* (for model providers): Our analysis clarifies the security boundaries of PRCs under concrete parameter choices, helping model providers deploy PRC-based watermarking more carefully and avoid insecure configurations. (2) *Guided Research Directions* (for researchers): Our findings may inspire future work on strengthening LDPC-PRC or on designing new PRC schemes based on alternative hardness assumptions, thereby advancing the robustness and reliability of watermarking research.

Negative Impacts: (1) *Potential Privacy Leakage* (for end users): The attacks described in this work could, in principle, be misused to detect PRC watermarks, potentially leading to privacy leakage. Currently, LDPC-PRC has not been widely deployed in LLMs due to limited error-correcting capability, so the immediate risk remains low. For diffusion-based generative models, improper parameter choices might allow adversaries to recover the secret key, compromising intended security guarantees. (2) *Perceived Confidence Reduction* (for researchers): Researchers may temporarily perceive LDPC-PRC as less secure after our analysis.

Mitigations. To mitigate these potential risks, providers of large generative models should carefully evaluate the security of LDPC-PRC under concrete parameter settings and apply the mitigations outlined in this work before any deployment. Responsible use entails rigorous parameter selection, comprehensive security analysis, and strict compliance with relevant legal and regulatory requirements.

Justifications. We argue that the benefits of this research outweigh the potential risks. PRC has been proposed as a foundational watermarking primitive, and it is therefore essential to conduct an independent security analysis before any widespread deployment. Our study demonstrates that the security of LDPC-PRC has been overestimated across all examined parameter regimes. In particular, embedding PRC watermarks into LLMs is impractical, and implementations on GIMs suffer from inappropriate parameter choices and incorrect key generation procedures. Our work highlights these issues to prevent incorrect deployments while preserving the security of the ecosystem.

Open Science

We release all source code and part of the experimental data at `https://github.com/1234wangtr/PRC_estimator` and permanently archived at `https://zenodo.org/records/20321793`. The accompanying `README.md` provides detailed instructions on estimating the complexity of our attacks under concrete parameters in Section 5, constructing the watermark schemes, and reproducing the real-world attacks described in Section 6.

References

[1] Scott Aaronson. My ai safety lecture for ut effective altruism, 2022. URL: `https://scottaaronson.blog/?p=6823`.

[2] Adil S. Al-Busaidi, Raghu Raman, Laurie Hughes, Mousa Ahmed Albashrawi, Tegwen Malik, Yogesh K. Dwivedi, Thuraiya Al- Alawi, Mohammed AlRizeiqi, Gareth Davies, Mark Fenwick, Parul Gupta, Shashikala Gurpur, Apeksha Hooda, Paulius Jurcys, Daryl Lim, Nicola Lucchi, Tanvi Misra, Ramakrishnan Raman, Anuragini Shirish, and Paul Walton. Redefining boundaries in innovation and knowledge domains: Investigating the impact of generative artificial intelligence on copyright and intellectual property rights. *Journal of Innovation & Knowledge*, 9(4):100630, 2024.

[3] Omar Alrabiah, Prabhanjan Ananth, Miranda Christ, Yevgeniy Dodis, and Sam Gunn. Ideal pseudorandom codes. In *Proceedings of the 57th Annual ACM Symposium on Theory of Computing*, pages 1638–1647, 2025.

[4] Bang An, Mucong Ding, Tahseen Rabbani, Aakriti Agrawal, Yuancheng Xu, Chenghao Deng, Sicheng Zhu, Abdirisak Mohamed, Yuxin Wen, Tom Goldstein, and Furong Huang. WAVES: benchmarking the robustness of image watermarks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[5] Anja Becker, Antoine Joux, Alexander May, and Alexander Meurer. Decoding random binary linear codes in $2^{n/20}$: How $1+1=0$ improves information set decoding. In *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 520–536. Springer, 2012.

[6] Elwyn Berlekamp, Robert McEliece, and Henk Van Tilborg. On the inherent intractability of certain

coding problems (corresp.). *IEEE Transactions on Information theory*, 24(3):384–386, 2003.

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

[8] Miranda Christ and Sam Gunn. Pseudorandom error-correcting codes. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024*, volume 14925 of *Lecture Notes in Computer Science*, pages 325–347, Santa Barbara, CA, USA, 2024. Springer.

[9] 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*, volume 247 of *Proceedings of Machine Learning Research*, pages 1125–1139. PMLR, 2024.

[10] Kaushik Deb, Md Sajib Al-Seraj, Md Moshiul Hoque, and Md Iqbal Hasan Sarkar. Combined dwt-dct based digital image watermarking technique for copyright protection. In *2012 7th International Conference on Electrical and Computer Engineering*, pages 458–461. IEEE, 2012.

[11] Ilya Dumer. On minimum distance decoding of linear codes. In *Proc. 5th Joint Soviet-Swedish Int. Workshop Inform. Theory*, pages 50–52. Moscow, 1991.

[12] Jaiden Fairoze, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Mingyuan Wang. Publicly-detectable watermarking for language models. *IACR Communications in Cryptology*, 1(4), 2025.

[13] Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. The stable signature: Rooting watermarks in latent diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22466–22477, 2023.

[14] Sanjam Garg, Sam Gunn, and Mingyuan Wang. Black-box crypto is useless for pseudorandom codes. In *Theory of Cryptography Conference*, pages 205–224. Springer, 2025.

[15] Surendra Ghentiyala and Venkatesan Guruswami. New constructions of pseudorandom codes. *arXiv preprint arXiv:2409.07580*, 2024.

[16] 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.

[17] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[18] Manav Gupta. Ai in social media: Statistics, trends driving content creation. URL: https://colorwhistle.com/ai-in-social-media/.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6 2016.

[20] Nikola Jovanović, Robin Staab, and Martin Vechev. Watermark stealing in large language models. In *ICML 2024*, 2024.

[21] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In *International Conference on Machine Learning*, pages 17061–17084. PMLR, 2023.

[22] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. On the reliability of watermarks for large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[23] Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

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

[25] Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip Yu. A survey of text watermarking in the era of large language models. *ACM Computing Surveys*, 57(2):1–36, 2024.

[26] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: a fast ode solver for diffusion probabilistic model sampling in around 10 steps. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc.

[27] Nicola Lucchi. Chatgpt: A case study on copyright challenges for generative artificial intelligence systems. *European Journal of Risk Regulation*, 15(3):602–624, 2024.

[28] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL: https://openreview.net/forum?id=rJzIBfZAb.

[29] Alexander May, Alexander Meurer, and Enrico Thomae. Decoding random linear codes in $\tilde{\mathcal{O}}(2^{0.054n})$. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 107–124. Springer, 2011.

[30] Alexander May and Ilya Ozerov. On computing nearest neighbors with applications to decoding of binary linear codes. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 203–228. Springer, 2015.

[31] OpenAI. gpt-3.5-turbo, 2023. URL: https://platform.openai.com/docs/models/gpt-3.5-turbo.

[32] OpenAI. gpt-4-turbo, 2023. URL: https://platform.openai.com/docs/models/gpt-4-turbo.

[33] OpenAI. gpt-4.1-turbo, 2025. URL: https://platform.openai.com/docs/models/gpt-4.1.

[34] OpenAI, Aaron Hurst, Adam Lerer, Adam P. Goucher, et al. Gpt-4o system card, 2024. arXiv:2410.21276.

[35] Leyi Pan, Aiwei Liu, Zhiwei He, Zitian Gao, Xuandong Zhao, Yijian Lu, Binglin Zhou, Shuliang Liu, Xuming Hu, Lijie Wen, et al. Markllm: An open-source toolkit for llm watermarking. *arXiv preprint arXiv:2405.10051*, 2024.

[36] Eugene Prange. The use of information sets in decoding cyclic codes. *IRE Transactions on Information Theory*, 8(5):5–9, 1962.

[37] Predis.ai. Ai ad generator. URL: https://predis.ai/.

[38] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10684–10695, 6 2022.

[39] Gustavo Santana. Gustavosta/stable-diffusion-prompts, 2022. URL: https://huggingface.co/datasets/Gustavosta/Stable-Diffusion-Prompts.

[40] Ryoma Sato, Yuki Takezawa, Han Bao, Kenta Niwa, and Makoto Yamada. Embarrassingly simple text watermarks. *arXiv preprint arXiv:2310.08920*, 2023.

[41] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[42] Jacques Stern. A method for finding codewords of small weight. In *International colloquium on coding theory and applications*, pages 106–113. Springer, 1988.

[43] Krti Tallam, John Kevin Cava, Caleb Geniesse, N Benjamin Erichson, and Michael W Mahoney. Removing watermarks with partial regeneration using semantic information. *arXiv preprint arXiv:2505.08234*, 2025.

[44] Matthew Tancik, Ben Mildenhall, and Ren Ng. Stegastamp: Invisible hyperlinks in physical photographs. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2117–2126, 2020.

[45] Qwen Team. Qwen3 technical report, 2025. URL: https://arxiv.org/abs/2505.09388, arXiv:2505.09388.

[46] Yuxin Wen, John Kirchenbauer, Jonas Geiping, and Tom Goldstein. Tree-rings watermarks: Invisible fingerprints for diffusion images. *Advances in Neural Information Processing Systems*, 36:58047–58063, 2023.

[47] Zijin Yang, Kai Zeng, Kejiang Chen, Han Fang, Weiming Zhang, and Nenghai Yu. Gaussian shading: Provable performance-lossless image watermarking for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12162–12171, 2024.

[48] Xuandong Zhao, Prabhanjan Vijendra Ananth, Lei Li, and Yu-Xiang Wang. Provable robust watermarking for AI-generated text. In *The Twelfth International Conference on Learning Representations*, 2024.

[49] Xuandong Zhao, Sam Gunn, Miranda Christ, Jaiden Fairoze, Andres Fabrega, Nicholas Carlini, Sanjam Garg, Sanghyun Hong, Milad Nasr, Florian Tramer, Somesh Jha, Lei Li, Yu-Xiang Wang, and Dawn Song. SoK: Watermarking for AI-Generated Content . In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 2621–2639, 2025.

[50] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 657–672, 2018.

Table 5: Key Mathematical Notations.

Notation	Description
λ	Security parameter
n	Code length of PRC
g	The column size of matrix $\mathbf{G}$ of PRC
k	The column size of matrix $\mathbf{G}$ of Π_{GIM}
t	Row-wise nonzero count of $\mathbf{P}$
r	The row size of secret key $\mathbf{P}$
$\mathbf{G}$	$\mathbf{G} \in \mathbb{F}_2^{n \times g}$ is a public generator matrix
$\mathbf{P}$	$\mathbf{P} \in \mathbb{F}_2^{r \times n}$ is a secret parity-check matrix
$\mathbf{z}$	$\mathbf{z} \in \mathbb{F}_2^n$ is a random vector serving as the one-time pad
pk	$\mathsf{pk} = (\mathbf{G}, \mathbf{z})$ is the public key of PRC
sk	$\mathsf{sk} = (\mathbf{P}, \mathbf{z})$ is the secret key of PRC
$w_H(\mathbf{y})$	Hamming weight of a vector $\mathbf{y}$
$\text{Ber}(p)$	Bernoulli distribution on $\{0,1\}$ with expectation $0 \leq p \leq 1$
$\text{Ber}(n,p)$	n-bit strings where each bit is an i.i.d sample from $\text{Ber}(p)$
$\langle \mathbf{x}, \mathbf{y} \rangle$	Component-wise product of $\mathbf{x}$ and $\mathbf{y}$

A Additional Details for PRC, LLM, and GIM

A.1 Mathematical Notations

The key mathematical notations are summarized in Table 5.

A.2 Detailed Algorithms for LLM and GIM

The detailed algorithms for LLM and GIM generation are provided in Algorithm 3 and Algorithm 4, respectively.

Algorithm 3 `LLM Generation`

Input: $\mathsf{prompt} \in \mathcal{T}^*$
Parameter: Temperature $\mathsf{temp} \in \mathbb{R}^+$, generation length $\mathsf{len} \in \mathbb{Z}^+$
Output: Generated text $T \in \mathcal{T}^*$

$T \leftarrow \perp$
for $i \leftarrow 0$ to $\mathsf{len} - 1$ **do**
  $\mathbf{l} \leftarrow \mathsf{LLM}(\mathsf{prompt} \parallel T)$ ▷ The logits of tokens
  $\mathbf{p} \leftarrow \mathsf{Softmax}(\mathbf{l}/\mathsf{temp})$ ▷ The probabilities of tokens
  $t \xleftarrow{\mathbf{p}} \mathcal{T}$ ▷ Sample the next token
  $T \leftarrow T \parallel t$
end for
return T

Algorithm 4 `GIM Generation`

Input: $\mathsf{prompt} \in \mathcal{T}^*$
Parameter: De-noising model `U-net`, Autoencoder model (`E-net`, `D-net`),
de-noising steps s, a (possibly randomized) de-noising scheduler $f(\cdot)$
Output: Generated image Img

$\mathbf{y}^s \xleftarrow{\$} \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$
for $i \leftarrow s$ down to 1 **do**
  $\mathbf{y}^{i-1} \leftarrow f(\texttt{U-net}(\mathsf{prompt}, \mathbf{y}^i, i), i)$
end for
$\mathsf{Img} \leftarrow \texttt{D-net}(\mathbf{y}^0)$
return Img

A.3 Detailed Algorithms for Π_{LLM} and Π_{GIM}

The specific algorithms involved in Π_{LLM} and Π_{GIM} are further described in Algorithm 5, Algorithm 6, and Algorithm 7.

Algorithm 5 `Token Sampling`

Input: Token probabilities $\mathbf{p}$
  codeword trunk $\mathbf{x}'$
Parameter: token space $\mathcal{T}$
Output: Sampled token $t \in \mathcal{T}$

$b \leftarrow \perp$
for $i \leftarrow 0$ to $\lceil \log_2 |\mathcal{T}| \rceil - 1$ **do**
  $p' \leftarrow \frac{\sum_{t \in \mathcal{T} \wedge \mathsf{Unpack}(t) = b \| 1 \|_*} p_t}{\sum_{t \in \mathcal{T} \wedge \mathsf{Unpack}(t) = b \|_*} p_t}$
    ▷ Marginal probability of current bit
  if $p' \leq 1/2$ **then**
    $t' \leftarrow \text{Ber}(2p'x'_i)$
  else
    $t' \leftarrow \text{Ber}(1 - 2(1-p')(1-x'_i))$
  end if
  $b \leftarrow b \parallel t'$ ▷ Sample token bit-wise
end for
$t \leftarrow \mathsf{Packbits}(b)$
return t

B Procedures of Attacks

This section describes the full procedures of the three attacks, which correspond to Attack-I, Attack-II, and Attack-III, respectively, as shown in Algorithm 8, Algorithm 9 and Algorithm 10.

C Complexities of Attacks

The detailed complexities of our attacks are summarized in Table 6 for Π_{LLM} and Table 7 for Π_{GIM}.

Table 6: Time complexities of all attacks on Π_{LLM}.

t	ε	ρ	$\log_2^{\mathsf{T}_{\text{partial}}}$	$\log_2^{\mathsf{P}_{\text{weak}}^{\text{LLM}}}$	$\log_2^{\mathsf{T}_{\text{dis}}}$	$\log_2^{\mathsf{T}_{\text{overlay}}}$	λ
3	0.236	0.264	21.30	0.00	22.58	72.21	48
4	0.285	0.215	29.19	−0.00	22.98	73.02	63
5	0.319	0.181	36.84	−3.91	27.20	73.50	78
6	0.344	0.156	44.28	−11.89	35.42	73.57	93
7	0.363	0.137	51.59	−19.66	43.39	73.61	107
8	0.377	0.123	58.76	−27.20	51.11	73.64	121
9	0.389	0.111	65.84	−34.55	58.62	73.66	135
10	0.399	0.101	72.82	−41.73	65.94	73.67	148
11	0.408	0.092	79.71	−48.75	73.08	73.54	162
12	0.415	0.085	86.54	−55.64	80.09	73.56	175
13	0.421	0.079	93.29	−62.40	86.95	73.46	188
14	0.426	0.074	99.99	−69.04	93.69	73.37	202

Table 7: Time complexities of all attacks on Π_{GIM}.

t	ε	ρ	η	$\log_2^{\mathsf{T}_{\text{partial}}}$	$\log_2^{\mathsf{P}_{\text{weak}}^{\text{GIM}}}$	$\log_2^{\mathsf{T}_{\text{dis}}}$	$\log_2^{\mathsf{T}_{\text{overlay}}}$	$\log_2^{\mathsf{T}'_{\text{overlay}}}$	λ
3	0.281	0.219	0.018	21.88	−0.01	23.16	244.03	56.56	39
4	0.325	0.175	0.013	27.92	−7.83	31.01	202.96	53.37	51
5	0.354	0.146	0.011	33.79	−16.71	39.92	176.52	51.40	63
6	0.375	0.125	0.009	39.52	−24.87	48.11	157.93	50.15	75
7	0.391	0.109	0.008	45.14	−32.61	55.87	144.27	49.22	86

Algorithm 6 `TextToCodeword`

Input: Text $T \in \mathcal{T}^{\mathsf{len}}$
Output: Recovered codeword $\mathbf{x}' \in \mathbb{R}^n$

$\mathbf{x}' = \perp$
for $i \leftarrow 0$ to $\mathsf{len}-1$ **do**
  $\mathbf{x}' \leftarrow \mathbf{x}' \,\|\, \mathsf{Unpackbits}(T_i)$
end for
return $\mathbf{x}'$

Algorithm 7 `ImageToCodeword`

Input: Image Img
Parameter: De-noising model `U-net`, Autoencoder model (`E-net`, `D-net`),
inversion steps s, a (possibly randomized) noise inversion scheduler $z(\cdot)$, error calibration factor σ
Output: Recovered codeword $\mathbf{x}' \in \mathbb{R}^n$

$\mathbf{y}^0 \leftarrow \texttt{E-net}(\mathsf{Img})$
for $i \leftarrow 0$ to $s-1$ **do**
  $\mathbf{y}^{i+1} \leftarrow z(\texttt{U-net}(\mathbf{y}^i, \perp, i+1), i+1)$
end for
$\mathbf{x}' = \mathsf{erf}(\mathbf{y}^s / \sqrt{2\sigma^2(1+\sigma^2)})$
return $\mathbf{x}'$

D Attack Performance Under Proxy Models

When performing our attacks under proxy models, we generate watermarked images with stable-diffusion-2-1-base, while extracting the codewords through Algorithm 7 using the autoencoders (`E-net`,`D-net`) from stable-diffusion-2-1-base (SD21), stable-diffusion-v1-5 (SD15), and stable-diffusion-2-base (SD20) [38]. Then, we calculate TPR values of $\text{TPR}_{\text{SD21}}, \text{TPR}_{\text{SD15}}$, and TPR_{SD20}. The parameter choice is the same as the formal set except that $\tau = 0.60$ for both Attack-I and Attack-II.

The results are presented in Table 8. We display only one FPR column, as the FPR is 0% for all vectors in the experiments. Our attack with proxy neural networks remains effective in both Attack-I and Attack-II. In particular, we observe that SD20 performs almost as well as the original target model SD21. These experimental results further demonstrate the practical threats posed by our proposed watermark removal attacks.

E Attack Performance Under Different Models

We conduct extra experiments using different generative models under the same setup as in Section 6.1 with $t =$

Algorithm 8 `Attack-I`

Input: Public Key $(\mathbf{G},\mathbf{z})$, target codewords $X = \{\mathbf{x}^{(1)},\cdots,\mathbf{x}^{(m)}\}$
Parameter: (α,β,r',l,τ) defined in Section 4.1 and (g,t,n,r) of LDPC-PRC
Output: true or false, where true indicates X are encoded by PRC and false otherwise

$L_1 \leftarrow \{(a_1,...,a_{t_1},\mathbf{sum}_1) | 1 \le a_1 < ... < a_{t_1} \le n, \mathbf{sum}_1 = \sum_{i=1}^{t_1} \mathbf{G}_{a_i}\}$
$L_2 \leftarrow \{(a_{t_1+1},...,a_t,\mathbf{sum}_2) | 1 \le a_{t_1+1} < ... < a_t \le n, \mathbf{sum}_2 = \sum_{i=t_1+1}^{t} \mathbf{G}_{a_i}\}$
if t is **even then**
    Construct $L_1' \subset L_1$ and $|L_1'| = \frac{1}{\sqrt{r'/l}}|L_1|$
    Construct $L_2' \subset L_2$ and $|L_2'| = \frac{1}{\sqrt{r'/l}}|L_2|$
else
    Construct $L_1' \subset L_1$ and $|L_1'| = \alpha|L_1|$
    Construct $L_2' \subset L_2$ and $|L_2'| = \beta|L_2|$
end if
Sort L_1', L_2'
$L' \leftarrow \text{MERGE-JOIN}(L_1', L_2')$
Initialize an empty list for vectors of V
for all $(a_1,\cdots,a_t) \in L'$ **do**
    $\mathbf{v} \leftarrow (v_1,\cdots,v_n)$ ▷ where $v_i = 1$ if $\exists j$ s.t. $i = a_j$ and $v_i = 0$ otherwise
    Append $\mathbf{v}$ to the list of vectors for V
end for
Initialize $N_{\mathsf{tot}}, N_{\mathsf{zero}} \leftarrow 0, 0$
for all $\mathbf{v}^{(j)} \in V$ and $\mathbf{x}^{(i)} \in X$ **do**
    $N_{\mathsf{tot}} \leftarrow N_{\mathsf{tot}} + 1$
    if $\langle \mathbf{v}^{(j)}, \mathbf{x}^{(i)} + \mathbf{z} \rangle = 0$ **then**
        $N_{\mathsf{zero}} \leftarrow N_{\mathsf{zero}} + 1$
    end if
end for
if $N_{\mathsf{zero}}/N_{\mathsf{tot}} \ge \tau$ **then**
    return true
else
    return false
end if

Table 8: Watermark detection attacks against real-world GIM with different proxy models when $t = 3$. The success rates of all attacks are 100%.

Attack	TPR_0	TPR_{SD21}	TPR_{SD15}	TPR_{SD20}	FPR
I	98%	95%	61%	95%	0%
II	100%	100%	95%	100%	0%

Algorithm 9 `Attack-II`

Input: Public Keys $\{(\mathbf{G}^{(i)},\mathbf{z}^{(i)})\}$, target codewords $X = \{\mathbf{x}^{(1)},\cdots,\mathbf{x}^{(m)}\}$ outputted by a certain oracle
Parameter: Threshold τ defined in Section 4.1 and (g,t,n,r) of LDPC-PRC
Output: true or false, where true indicates X are generated by LDPC-PRC and false otherwise

Randomly select a public key pair $(\mathbf{G},\mathbf{z}) \xleftarrow{\$} \{(\mathbf{G}^{(i)},\mathbf{z}^{(i)})\}$
$I \leftarrow \{(\alpha_j,\beta_j) | \mathbf{G}_{\alpha_j} = \mathbf{G}_{\beta_j}\}$
if $|I| = 0$ **then**
    goto line 1
end if
Obtain X from the oracle corresponding to $(\mathbf{G},\mathbf{z})$
Initialize $N_{\mathsf{tot}}, N_{\mathsf{dup}} \leftarrow 0, 0$
for all $(\alpha_j,\beta_j) \in I$ and $\mathbf{x}^{(i)} \in X$ **do**
    $N_{\mathsf{tot}} \leftarrow N_{\mathsf{tot}} + 1$
    if $x_{\alpha_j}^{(i)} = x_{\beta_j}^{(i)}$ **then**
        $N_{\mathsf{dup}} \leftarrow N_{\mathsf{dup}} + 1$
    end if
end for
if $N_{\mathsf{dup}}/N_{\mathsf{tot}} \ge \tau$ **then**
    return true
else
    return false
end if

Algorithm 10 `Attack-III`

Input: Public Key $(\mathbf{G},\mathbf{z})$, target codeword $\mathbf{x} = \mathbf{Gs} + \mathbf{e} + \mathbf{z}$
Parameter: noise rate μ defined in Section 4.3 and (g,t,n,r) of LDPC-PRC
Output: $\mathbf{x}'' = \mathbf{x} + \mathbf{e}'$ s.t. $\text{Decode}(1^\lambda, \mathsf{sk}, \mathbf{x}'') = \perp$ and $w_H(\mathbf{e}') = \mu n$

$\mathbf{x}' \leftarrow \mathbf{x} + \mathbf{z}$
$\mathbf{e} \leftarrow \text{ISD}(\mathbf{G},\mathbf{x}')$ ▷ recover $\mathbf{e}$ from $\mathbf{x}', \mathbf{G}$ with an ISD algorithm
$S \leftarrow \{1,\cdots,n\} \backslash \text{supp}(\mathbf{e})$ ▷ where $\text{supp}(\mathbf{e})$ denotes the support set of $\mathbf{e}$
Choose $T \subset S$ and $|T| = \mu n$
Construct $\mathbf{e}'$ with support set T
$\mathbf{x}'' \leftarrow \mathbf{x} + \mathbf{e}'$
return $\mathbf{x}''$

3. We employ Qwen-3B [45] for Π_{LLM}, where Attack-I achieves $\text{TPR}_0 = 99\%, \text{TPR}_1 = 33\%, \text{FPR} = 1\%$ and Attack-II achieves $\text{TPR}_0 = 100\%, \text{TPR}_1 = 100\%, \text{FPR} = 0\%$. It is worth noting that Π_{LLM} also exhibits impractical quality on Qwen-3B. For Π_{GIM}, we utilize stable-diffusion-v1-5 [38], where $\text{TPR}_0 = 98\%, \text{TPR}_1 = 84\%, \text{FPR} = 2\%$ and Attack-II achieves $\text{TPR}_0 = 100\%, \text{TPR}_1 = 97\%, \text{FPR} = 0\%$. In all cases, the success rate is 100%. The gap between TPR and FPR demonstrates that our distinguishing attacks remain effective across diverse models.

F Artifact Appendix

F.1 Abstract

Pseudorandom error-correcting codes (PRCs), a novel cryptographic primitive recently proposed at CRYPTO 2024, are primarily applied in undetectable watermarking schemes for large generative models. However, the security of PRCs has not yet been systematically analyzed. To fill this gap, we present the first cryptanalysis of PRCs. Specifically, focusing on LDPC-PRC, the only known practical instantiation of PRCs, we propose three novel attacks that challenge its undetectability (Attack I, II) and robustness (Attack III). To rigorously demonstrate the practical threat, this artifact analyzes the concrete attack complexity under realistic parameters and validates the attack effectiveness on both real-world large language models and generative image models, including DeepSeek and Stable Diffusion. Our analysis shows that the claimed security guarantees of LDPC-PRC are undermined across all practically feasible regimes.

F.2 Description & Requirements

F.2.1 Security, privacy, and ethical concerns

The artifact attacks the PRC watermark scheme deployed in self-hosted AIGC systems without involving any local data, real-world systems or human subjects. Therefore, it does not raise security, privacy, or ethical concerns.

F.2.2 How to access

Our artifact is permanently and publicly available at `https://zenodo.org/records/21771706` with DOI:10.5281/zenodo.21771706. Our artifact can also be downloaded from GitHub with:

```
git clone https://github.com/1234wangtr/PRC_estimator
```

F.2.3 Hardware dependencies

Fully running this artifact requires a GPU machine with at least 80GB VRAM (Ours: NVIDIA H20 with 141GB of VRAM), equipped with a CPU with moderate computational power (Ours: Intel XEON PLATINUM 8558P 2.70 GHz), 16GB RAM (Ours: 2TB), and 60GB free disk space. However, the main claims can be verified on a CPU-only machine with 5 GB of free disk space, given pre-generated data.

F.2.4 Software dependencies

OS: Modern x86_64 Linux with `git`, `bash`, and `unzip` installed (Ours: Ubuntu 24.04.1, Kernel 6.17.0-29-generic).

Python: 3.11, with `pip` installed and supports `conda` virtual environments. (Ours: 3.11.4 with Miniconda 25.3.1)

(Optional for GPU) NVIDIA Driver: 525.60.13+ to support CUDA 12.x. (Ours: 595.71.05, CUDA 12.1.105)

(Optional) Fonts: *Times New Roman* and *Droid Sans Fallback*. You can install it via `setup/get_fonts.sh`.

F.2.5 Benchmarks

None. This artifact does not involve third-party benchmarks.

F.3 Set-up

F.3.1 Installation

After downloading the artifact, you can enter `PRC_estimator` folder, which will be the working directory for all the following commands.

Our artifact involves two separate Python virtual environments for LLM and GIM experiments. You can set them up with: `conda env create -f llm/environment.yml && conda env create -f gim/environment.yml`

F.3.2 Basic Test

You can run Experiment 1 (E1) to test if the environments are correctly set up.

F.4 Evaluation workflow

F.4.1 Major Claims

(C1): Our attacks reveal that PRC-based watermarking schemes failed to achieve 128-bit security under realistic parameters. This is proven by the experiment E1 described in Section 5.2 of our paper, whose results are presented in Figure 4 and Tables 6 and 7 of our paper.

(C2): Our Attacks I and II can successfully distinguish real-world LDPC-PRC watermarks for generated text. This is proven by the experiment E2 described in Section 6.2.2 and Section 6.2.3 of our paper, whose results are presented in Table 3 of our paper.

(C3): Employing LDPC-PRC watermarks for text generation only works for large temperatures, which leads to a significant quality degradation.

(C4): Our Attacks I and II can successfully distinguish real-world LDPC-PRC watermarks for generated images. This is proven by the experiment E4 described in Section 6.3.2 of our paper, whose results are presented in Table 4 of our paper. These attacks can success even when the adversary has access only to a proxy model.

This is proven by the experiment E4 described in Appendix D of our paper, whose results are presented in Table 8 of our paper.

(C5): The attacker can perform Attack III to remove the LDPC-PRC watermarks for images with a high success rate under a reasonable image quality budget. This is proven by the experiment E5 described in Section 6.3.2 of our paper, whose results are presented at the end of Section 6.3.2 of our paper.

F.4.2 Experiments

(E1): Concrete Time Complexity Analysis [1 human-minute + 1 CPU minute]. Estimate the concrete time complexity of our attacks against the LDPC-PRC schemes with LLM and GIM watermarking parameters.
Execution: Run the security estimation scripts as:

```
conda run -n prc-estimator-llm \
  python llm/security_estim/main.py
conda run -n prc-estimator-gim \
  python gim/security_estim/main.py
```

Results: The figures of attack time complexity are in `<llm or gim>/data/security_estim.pdf`, corresponding to Figure 4 (a) and (b). The detailed results are in `<llm or gim>/data/security_estim.csv`, corresponding to Tables 6 and 7.

(E2): Attack I and II against LDPC-PRC Watermarked Text [1 human-minute + 3 CPU-minutes/10 groups]. Perform Attacks I and II against the LDPC-PRC watermarked text generated by DeepSeek-R1-Distill-Qwen-7B under $t = 3$ and temperature 1.8.
Using pre-generated victim data: Since generating watermarked text requires significant GPU resources, we provide 10 groups of pre-generated watermarked text under $t = 3$. You can obtain the data by:

```
unzip llm/data/Deepseek_t_3_temp_1.8.zip \
  -d llm/data/Deepseek_t_3_temp_1.8_example
```

(Optional) Self-generating victim data: [Extra 5 GPU-hours and 0.2GB disk space for 10 groups, extra 16GB disk space for model] First, download the model using `setup/get_llm.sh`. Then, generate watermarked text with:

```
conda activate prc-estimator-llm
export CUDA_VISIBLE_DEVICES=0
python llm/generation/main.py \
  --temperature 1.8  --model_name Deepseek \
  --prc_t 3 --start 0 --end 10
```

This will generate 10 groups of watermarked text with different pk to `llm/data/Deepseek_t_3_temp_1.8`.
Execution: The attacks can be performed with:

```
conda activate prc-estimator-llm
python llm/attack/attack1_2_main.py --t 3 \
  llm/data/Deepseek_t_3_temp_1.8[_example]
```

This attack aims to distinguish PRC codewords and random vectors.
Results: The attack results are in `llm/data/Deepseek_t_3_temp_1.8[_example]/result.csv`, corresponding to Table 3. The meaning of each field is as follows:

- **$\mathbf{TPR_0}$:** The true-positive rate at which PRC codewords are correctly classified (higher is better).
- **$\mathbf{FPR_0}$:** The false-positive rate at which random vectors are incorrectly classified as PRC codewords (lower is better).
- **$\mathbf{TPR_1}$:** Measures whether the inversed PRC codeword can be detected as a watermark. A significant gap between TPR_1 and FPR_0 demonstrates better effectiveness of the attack.
- **Success Rate:** For Attack-I, it corresponds to the proportion of instances in which at least one row of the secret key is successfully recovered. For Attack-II, it corresponds to the proportion of instances in which at least one pair of identical rows is successfully identified.

Although our paper reports results over 128 groups, the results over 10 groups should have the same trend.
The attack details are in `llm/data/Deepseek_t_3_temp_1.8[_example]/log.txt`. You can inspect some intermediate outputs such as:

```
# Recovered nonzero positions of the secret key
grep "found l1_vec" \
llm/data/Deepseek_t_3_temp_1.8[_example]/log.txt
# Duplicated positions identified from pub key
grep "dup_dict" \
llm/data/Deepseek_t_3_temp_1.8[_example]/log.txt
```

(E3): Watermarked Text Quality Analysis [2 human-minutes + 2 CPU minutes]. Analyze the pre-generated quality of the generated watermarked text.
Preparation: Since the quality analysis requires lots of watermarked text generated under different temperatures, the analysis is performed on the pre-generated data, which can be obtained by:

```
unzip -d llm/data \
  llm/data/Deepseek_t_3_temp_all.zip
```

Execution: Run the following command to analyze the quality of the generated watermarked text.

```
conda run -n prc-estimator-llm \
  python llm/generation/plot_entropy.py
```

Results: The generated figure is in `llm/data/avg_entropy_vs_correct_rate.pdf`, corresponding to Figure 5.

(E4): Attack I and II against LDPC-PRC Watermarked Image [1 human-minute + 6 CPU-minutes/10 groups]. Perform Attacks I and II against the LDPC-PRC watermarked image generated under $t = 3$.
Using pre-generated victim data: Since generating watermarked images requires significant GPU resources, we provide 10 groups of pre-generated watermarked images under $t = 3$. You can obtain the data by:

```
unzip gim/data/SD21_t3.zip \
  -d gim/data/SD21_t3_example
unzip gim/data/SD21_t3_inv_SD15_SD2_SD21.zip \
  -d gim/data/SD21_t3_inv_SD15_SD2_SD21_example
```

(Optional) Self-generating victim data: [Extra 2 GPU-hour, 1GB disk space for 10 groups, extra 25GB disk space for model] First, download the model and dataset with `setup/get_gim.sh`. Then, generate watermarked images with:

```
conda activate prc-estimator-gim
export CUDA_VISIBLE_DEVICES=0
python gim/generation/main.py \
  --prc_t 3 --start 0 --end 10 \
  --gen_model_id SD21 --inv_model_ids SD21
# using proxy models
python gim/generation/main.py \
  --start 0 --end 10 --gen_model_id SD21 \
  --inv_model_ids SD15,SD2,SD21 --prc_t 3
```

This will generate 10 groups of watermarked images with different pk to `gim/data/SD21_t3`, along with 10 groups of watermarked images inverted by 3 different proxy models to `gim/data/SD21_t3_inv_SD15_SD2_SD21`

Execution: The attacks can be performed with:

```
conda activate prc-estimator-gim
python gim/attack/attack1_2_main.py --t 3 \
  gim/data/SD21_t3[_example]
python gim/attack/attack1_2_diff_inv_main.py \
  gim/data/SD21_t3_inv_SD15_SD2_SD21[_example] \
  --t 3 # using proxy models
```

This attack aims to distinguish PRC codewords and random vectors.

Results: The attack results without using proxy model are in `gim/data/SD21_t3[_example]/result.csv`, corresponding to Table 4. The attack results with proxy model are in `gim/data/SD21_t3_inv_SD15_SD2_SD21[_example]/result.csv`, corresponding to Table 8. The meaning of these results are the same as described in (E2). Although our paper reports results over 128 groups, the results over 10 groups should have the same trend. You can inspect some intermediate outputs at the corresponding `log.txt` files as demonstrated in (E2).

(E5): Watermark Removal Analysis [5 human-minutes + 2 GPU-hour for 10 images].

Preparation: Make sure the model and dataset are downloaded with `setup/get_gim.sh`.

Execution: First, generate watermarked images with:

```
conda activate prc-estimator-gim
export CUDA_VISIBLE_DEVICES=0
python gim/generation/main-for-attack3.py \
  --prc_t 3 --start 0 --end 10 \
  --model_id SD21
```

This will generate 10 watermarked images and save intermediate results to `gim/data/attack3_SD21_t3` in 2 GPU-minutes.

Then, generate the adversarial images with $\varepsilon = 16.0$ based on those intermediate results (corresponding to the attack results for Attack III) with:

```
conda activate prc-estimator-gim
export CUDA_VISIBLE_DEVICES=0
python gim/attack/attack3_main.py \
  gim/data/attack3_SD21_t3 --start 0 --end 10 \
  --model_id SD21 --eps 16.0
```

The attack will run for 1 GPU-hour for 10 images.

Results: The generated adversarial images are in `gim/data/attack3_SD21_t3/inv_lat_16.0`, and the images with removed watermarks are in `gim/data/attack3_SD21_t3/adv_img_16.0`.

To calculate the removal success rate, run:

```
conda activate prc-estimator-gim
python gim/attack/attack3_stati.py \
  gim/data/attack3_SD21_t3/inv_lat_16.0
```

This result corresponds to the attack success rate reported at the end of Section 6.3.2.

F.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2026/.