

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Cryptanalysis of LDPC-Based Pseudorandom Error-Correcting Codes

Tianrui Wang, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University;* Anyu Wang, *Institute for Advanced Study and State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University; Zhongguancun Laboratory;* Tianshuo Cong, *School of Cryptologic Science and Engineering and Shandong Key Laboratory of Artificial Intelligence Security, Shandong University;* Delong Ran, *Institute for Network Sciences and Cyberspace, BNRist, Tsinghua University;* Jinyuan Liu, *Institute for Advanced Study, Tsinghua University;* Xiaoyun Wang, *Institute for Advanced Study and State Key Laboratory of Cryptography and Digital Economy Security, Tsinghua University; Zhongguancun Laboratory; National Financial Cryptography Research Center; Shandong Institute of Blockchain*

https://www.usenix.org/conference/usenixsecurity26/presentation/wang-tianrui

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: 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*

A Artifact Appendix

A.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.

A.2 Description & Requirements

A.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.

A.2.2 How to access

Our artifact is permanently and publicly available at https://zenodo.org/records/21771706 with DOI https://doi.org/10.5281/zenodo.21771706.

Our artifact can also be downloaded from GitHub with:
`git clone https://github.com/1234wangtr/PRC_estimator`

A.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.

A.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`.

A.2.5 Benchmarks

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

A.3 Set-up

A.3.1 Installation

After downloading the artifact, you can enter `PRC_estimator` folder, which will be the working directory for all the follow-

ing 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`

A.3.2 Basic Test

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

A.4 Evaluation workflow

A.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.

A.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:

- **TPR_0:** The true-positive rate at which PRC codewords are correctly classified (higher is better).
- **FPR_0:** The false-positive rate at which random vectors are incorrectly classified as PRC codewords (lower is better).
- **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.

A.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/.