Shadowmark — Demo Report

Project: Multimedia Data Security — Capture the Mark

Group: shadowmark

Date: 2025

---

1. Introduction

This demo describes Shadowmark: a reproducible pipeline to embed a robust spread-spectrum watermark in grayscale images (DCT domain), detect its presence using a "centering-based" residual-ratio feature, compute a detection threshold tau from a ROC curve, and evaluate attacks.

The main idea has been refined for robust detection. We embed in selected DCT mid-bands using adaptive strength. For detection, we extract three ratio vectors: from the watermarked image (v_{wm}), the attacked image (v_{att}), and the original image (v_{clean}). We then compute **centered features**:

- v_{wm_c} = v_{wm} - v_{clean} (the "pure" watermark signal)
- v_{att_c} = v_{att} - v_{clean} (the "remaining" signal)

The final similarity is a weighted cosine comparison between v_{wm_c} and the calibrated v_{att_c}, checked against a ROC-derived global threshold tau.

Files included for submission:

- embedding.py — Spread-spectrum embedding.
- detection_shadowmark.py — Detector based on the centered-feature logic.
- roc_threshold.py — ROC computation and \tau selection.
- attacks.py — Single attacks and combined strategies.

Project assets (not for submission, but required to run):

- shadowmark.npy — The official watermark key.
- images/ — The dataset folder (e.g., 0000.bmp … 0100.bmp).

---

2. Requirements & Environment

- **Python:** 3.8–3.10
- **Setup:**

Bash

```
# Create virtual environment
```

```
python -m venv OFF_MDS_ENV


# Activate (Windows PowerShell)

# .\OFF_MDS_ENV\Scripts\Activate.ps1


# Activate (macOS / Linux)

# source OFF_MDS_ENV/bin/activate


# Install requirements

pip install -r requirements.txt

pip install ldpc
```

- **Quick Check:**

Bash

```
python -c "import numpy, cv2, scipy, sklearn, matplotlib; print('libs ok')"
```

---

3. Key Detection Enhancements

The detection pipeline was significantly reworked to **eliminate false positives** (especially on original, unwatermarked images) and improve robustness.

- **Feature Centering:** This is the most critical change. The detector now subtracts the original image's feature vector (v_clean) from both the watermarked (v_wm) and attacked (v_att) vectors. This isolates the pure watermark signal. When testing an original image, v_{att_c} becomes a near-zero vector, causing the similarity score to drop to zero and eliminating the false positive.
- **Prefilter:** A (5x5) Gaussian blur (sigma 1.0) is applied before DCT to stabilize features against noise and compression artifacts.
- **Winsorization:** Clips outlier values in DCT ratio vectors (p=2%) to reduce the influence of extreme coefficients.
- **Per-band Calibration:** Applies a linear gain calibration to the centered-attacked vector (v_{att_c}) per frequency band (low/mid/high). Gain is clipped to a safe interval (0.4, 3.0).
- **Weighted Cosine Similarity:** Uses weights (1.85 / 1.15 / 0.30) to give higher importance to the more stable low/mid-frequency bands.

- **WPSNR Reject (Removed):** The preliminary WPSNR < 25 dB reject **has been removed** to comply with challenge rules. The detector logic runs on all images, and WPSNR_REJECT is set to 0.00.
- **ROC Recalibration:** The threshold tau is computed by roc_threshold.py using this exact centering pipeline to target an FPR < 5%.

---

4. Quick File Summary

- embedding.py

Embeds the watermark in DCT coordinates using adaptive strength (ALPHA=0.017) and spatial redundancy (REDUNDANCY_FACTOR=2) for robustness.

  - **API:** embedding(path, watermark) -> np.uint8 (returns image, no file I/O).
- detection_shadowmark.py

Implements the non-blind detector.

  - **API:** detection(original_path, watermarked_path, test_path) -> (presence:int, wpsnr:float)
  - **Internals:**
    1. Loads the fixed global threshold tau (e.g., 0.436230).
    2. Extracts centered features: v_{wm_c} = R(wm) - R(clean) and v_{att_c} = R(att) - R(clean).
    3. Applies winsorization and per-band calibration to v_{att_c}.
    4. Computes the weighted cosine similarity sim.
    5. Performs a random-baseline safety check (sim > max_rnd + MIN_MARGIN).
    6. Returns presence = 1 if sim >= tau and the safety check passes.
- attacks.py

Provides implementations for all allowed attacks: JPEG, AWGN, blur, median, resize, and sharp. It also includes multi-step strategies and optimized combo helpers (e.g., attack_resize_jpeg_median) used in the attack phase.

- roc_threshold.py

Generates the ROC curve and selects the optimal tau. It uses the identical feature centering pipeline (extract, center, calibrate, score) as the detector. It generates positive scores (from light attacks) and negative scores (from clean images) to select the tau that meets the TARGET_FPR = 0.05 requirement.

---

5. Demo: Step-by-Step Commands

Run all commands from the project root.

5.1 Compute ROC and estimate tau (requires images/ dataset)

Bash

```
python roc_threshold.py
```

- **Produces:** roc.png, roc.pdf, tau.json, threshold.txt.
- Example result from our final run:

```
{ "tau": 0.436230, "AUC": 1.000, "FPR": 0.000, "TPR": 1.000 }
```

5.2 Create a sample watermarked image (Lena)

Bash

```
python - <<'PY'

from embedding import embedding

import cv2

wm = embedding("lena_grey.bmp", "shadowmark.npy")

cv2.imwrite("shadowmark_lena_grey.bmp", wm)

print("Saved: shadowmark_lena_grey.bmp")

PY
```

5.3 Self-detection check (watermark vs itself)

Bash

```
python - <<'PY'

from detection_shadowmark import detection

p, w = detection("lena_grey.bmp", "shadowmark_lena_grey.bmp",
"shadowmark_lena_grey.bmp")

print("presence:", p, "wpsnr:", w)

PY
```

- **Expected:** presence: 1. (WPSNR will be a very large sentinel value).

5.4 Valid destroy attack example (JPEG qf=40)

A known combination that yields presence=0 while keeping WPSNR >= 35.

Bash

```
python - <<'PY'

import cv2

from attacks import attack_jpeg

from detection_shadowmark import detection


wm = cv2.imread("shadowmark_lena_grey.bmp", 0)

att = attack_jpeg(wm, qf=40)

cv2.imwrite("shadowmark_lena_grey_jpeg40.bmp", att)


p,w = detection("lena_grey.bmp", "shadowmark_lena_grey.bmp",
"shadowmark_lena_grey_jpeg40.bmp")

print("jpeg40 -> presence:", p, "wpsnr:", w)

PY
```

- **Acceptable challenge result:** presence: 0 and WPSNR >= 35 dB.

---

6. Interpretation of Results

- presence == 1 à Detector believes the watermark is present (similarity >= tau).
- presence == 0 à Detector did not find the watermark (similarity tau).
- WPSNR quantifies perceptual quality. Higher is better.
- **A valid attack** requires presence == 0 and WPSNR >= 35 dB.