Hướng Dẫn Thực Hành: Echo Hiding Time Spread

Tháng 5, 2025

1 Lý thuyết về Echo Hiding Time Spread

Kỹ thuật **Echo Hiding Time Spread** là một phương pháp ẩn thông tin (steganography) trong tín hiệu âm thanh bằng cách thêm các tiếng vọng (echo) với các độ trễ khác nhau để mã hóa thông điệp. Phương pháp này tận dụng đặc điểm của hệ thống thính giác con người, vốn ít nhạy cảm với các thay đổi nhỏ trong tín hiệu âm thanh do echo gây ra, để ẩn dữ liệu mà không làm ảnh hưởng đáng kể đến chất lượng âm thanh.

1.1 Nguyên lý hoạt động

Trong kỹ thuật Echo Hiding Time Spread, mỗi bit của thông điệp (0 hoặc 1) được mã hóa bằng cách thêm một echo vào tín hiệu âm thanh gốc với độ trễ cụ thể:

- **Bit 0**: Thêm echo với độ trễ δ_0 (ví dụ: 200 mẫu ở tần số lấy mẫu 44100 Hz).
- **Bit 1**: Thêm echo với độ trễ δ_1 (ví dụ: 300 mẫu).

Echo được thêm với biên độ nhỏ (thường được điều chỉnh bởi hệ số α, ví dụ: $\alpha = 0.6$) để đảm bảo không gây biến dạng rõ rệt. Tín hiệu kết quả có dạng:

$$y(n) = x(n) + \alpha \cdot x(n - \delta_k)$$

trong đó:

- $x(n)$: Tín hiệu âm thanh gốc.
- $y(n)$: Tín hiệu âm thanh đã được nhúng thông điệp.
- δ_k: Độ trễ tương ứng với bit ($k = 0$ hoặc $k = 1$).
- α: Biên độ echo (hệ số suy giảm).

1.2 Time Spread

Điểm đặc biệt của kỹ thuật Time Spread là cách tổ chức các khung thời gian (frame) để nhúng thông điệp. Mỗi khung có độ dài cố định (ví dụ: 8192 mẫu) và chứa một bit thông điệp. Các khung được trải đều trên toàn bộ tín hiệu âm thanh, đảm bảo rằng thông điệp được phân bố đồng đều trong thời gian. Điều này giúp tăng khả năng chống nhiễu và giảm nguy cơ phát hiện thông điệp ẩn bằng các phương pháp phân tích tín hiệu đơn giản.

Để tăng hiệu quả, một cửa sổ mượt (ví dụ: cửa sổ Hanning) thường được áp dụng cho các khung để giảm hiện tượng gián đoạn ở ranh giới khung, giúp tín hiệu âm thanh nghe tự nhiên hơn. Công thức tín hiệu nhúng có thể được biểu diễn:

$$y(n) = x(n) + m(n) \cdot \alpha \cdot x(n - \delta_k) \cdot w(n)$$

trong đó:

- $m(n)$: Tín hiệu điều chế (1 cho bit 1, 0 cho bit 0).
- $w(n)$: Cửa sổ mượt (Hanning).

1.3 Ưu điểm và hạn chế

Ưu điểm:

- Ít làm thay đổi chất lượng âm thanh, phù hợp với các ứng dụng yêu cầu tính bí mật cao.
- Dễ triển khai với các tham số điều chỉnh linh hoạt (δ_0, δ_1, α, độ dài khung).
- Khả năng chống nhiễu tốt nhờ phân bố thời gian (time spread).

Hạn chế:

- Dung lượng thông điệp bị giới hạn bởi độ dài tín hiệu âm thanh và số khung.
- Yêu cầu các tham số mã hóa và giải mã phải được đồng bộ chính xác.
- Nhạy cảm với nhiễu mạnh hoặc các kỹ thuật xử lý âm thanh như nén mất dữ liệu.

Phần thực hành này sẽ áp dụng kỹ thuật Echo Hiding Time Spread để nhúng thông điệp vào tín hiệu âm thanh và chuẩn bị cho việc giải mã trong các bài lab tiếp theo.

2 Giới thiệu

Bài thực hành này giới thiệu về kỹ thuật Echo Hiding Steganography, một phương pháp ẩn thông điệp văn bản trong tín hiệu âm thanh bằng cách thêm các hiệu ứng echo với độ trễ khác nhau. Dựa trên lý thuyết Time Spread đã trình bày, bài thực hành bao gồm ba tác vụ chính, được triển khai bằng các script Python và thực hiện trong môi trường Labtainer:

1. **Tác vụ 1: Tạo tín hiệu đầu vào** - Tạo một file âm thanh sóng sin sạch.
2. **Tác vụ 2: Tiền xử lý âm thanh** - Chuẩn hóa và chuyển đổi âm thanh thành mảng NumPy.
3. **Tác vụ 3: Nhúng thông điệp** - Mã hóa thông điệp văn bản vào âm thanh bằng kỹ thuật Echo Hiding Time Spread.

Mục tiêu là nhúng thông điệp "A" vào tín hiệu âm thanh và chuẩn bị cho việc giải mã trong các bài thực hành tiếp theo. Môi trường Labtainer đảm bảo tính nhất quán và an toàn cho bài thực hành.

3 Yêu cầu

- **Labtainer**: Đã cài đặt trên hệ thống (Ubuntu, macOS, hoặc Windows với WSL2). Xem hướng dẫn cài đặt tại: https://my.nps.edu/web/c3o/labtainers.
- **Docker**: Đảm bảo Docker được cài đặt và đang chạy.
- Kiến thức cơ bản về Python và dòng lệnh.
- Quyền truy cập vào các file bài thực hành, bao gồm các script Python cho từng tác vụ.

4 Thiết lập bài thực hành

Bài thực hành Labtainer được cấu hình như sau:

- **Container**: Ubuntu 20.04 (`labtainer.ubuntu20`).
- **Phụ thuộc**: Python 3, NumPy, SciPy.
- **Script**:
  - `generate_input_audio.py` (Tác vụ 1)
  - `preprocess.py` (Tác vụ 2)
  - `embed.py` (Tác vụ 3)

Để bắt đầu bài thực hành:

1. Mở terminal và chuyển đến thư mục bài thực hành.
2. Chạy bài thực hành Labtainer:

```
labtainer time_spread_encoding -r
```

3. Lệnh này khởi động một container Docker chứa các script và phụ thuộc cần thiết.

5 Hướng dẫn từng tác vụ

5.1 Tác vụ 1: Tạo tín hiệu đầu vào

Mục tiêu: Tạo một file âm thanh sóng sin 2 giây, tần số 440 Hz (`input.wav`) để sử dụng trong các tác vụ tiếp theo.
Script: `generate_input_audio.py`
Các bước:

1. Chuyển đến thư mục bài thực hành trong container Labtainer.
2. Chạy script để tạo `input.wav`:

```
python3 generate_input_audio.py input.wav --duration 2.0
   --frequency 440.0 --amplitude 0.5
```

3. **Tham số**:
    - `–duration 2.0`: Độ dài 2 giây (đảm bảo đủ khung để nhúng thông điệp).
    - `–frequency 440.0`: Sóng sin tần số 440 Hz (nốt A4).
    - `–amplitude 0.5`: Biên độ bằng nửa giá trị tối đa để tránh clipping.
4. Kiểm tra đầu ra:
    - Đảm bảo file `input.wav` đã được tạo trong thư mục hiện tại.
    - Đặc điểm file: WAV 16-bit, 44100 Hz, mono, khoảng 2 giây.

Lưu ý: Sóng sin được chọn vì tính đơn giản, giúp việc phát hiện echo dễ dàng hơn. Đảm bảo file được tạo trước khi chuyển sang Tác vụ 2.

5.2 Tác vụ 2: Tiền xử lý âm thanh

Mục tiêu: Chuẩn hóa file âm thanh đầu vào và lưu dưới dạng mảng NumPy (`preprocessed.npy`) để nhúng thông điệp.
Script: `preprocess.py`
Các bước:

1. Đảm bảo `input.wav` từ Tác vụ 1 đã có sẵn.
2. Chạy script tiền xử lý:

```
python3 preprocess.py input.wav preprocessed.npy
```

3. **Quá trình**:
    - Script đọc `input.wav` và kiểm tra định dạng WAV 16-bit.
    - Âm thanh được chuẩn hóa (chia cho biên độ tối đa) và chuyển sang kiểu float32.

- Âm thanh mono được định dạng lại thành mảng 2D (N, 1) để tương thích với Tác vụ 3.
- Kết quả được lưu dưới dạng `preprocessed.npy`.

4. Kiểm tra đầu ra:
   - Đảm bảo file `preprocessed.npy` đã được tạo.
   - Định dạng mong đợi: Mảng NumPy, kích thước (N, 1), giá trị float32 trong [-1, 1].

Lưu ý: Nếu gặp lỗi "FileNotFoundError", kiểm tra xem `input.wav` đã được tạo đúng. Nếu file không phải WAV 16-bit, script sẽ báo lỗi "ValueError".

5.3 Tác vụ 3: Nhúng thông điệp

Mục tiêu: Nhúng thông điệp văn bản "A" vào âm thanh đã tiền xử lý bằng Echo Hiding Time Spread và lưu kết quả thành `output.wav`.

Script: `embed.py`

Các bước:

1. Đảm bảo `preprocessed.npy` từ Tác vụ 2 đã có sẵn.
2. Chạy script nhúng thông điệp:

```
python3 embed.py preprocessed.npy output.wav "A" --rate 44100
```

3. **Tham số**:
   - `preprocessed.npy`: Mảng NumPy đầu vào từ Tác vụ 2.
   - `output.wav`: File WAV đầu ra chứa thông điệp ẩn.
   - `"A"`: Thông điệp cần nhúng (ASCII 65, nhị phân 01000001).
   - `--rate 44100`: Tần số lấy mẫu (phải khớp với Tác vụ 1).
4. **Quá trình**:
   - Script tải `preprocessed.npy` và kiểm tra định dạng 2D.
   - Thông điệp "A" được chuyển thành chuỗi nhị phân (01000001).
   - Mỗi bit được nhúng vào một khung 8192 mẫu bằng echo với độ trễ `d0=200` (bit 0) hoặc `d1=300` (bit 1), biên độ `alpha=0.6`.
   - Cửa sổ Hanning được áp dụng để mượt hóa tín hiệu.
   - Đầu ra được chuẩn hóa và lưu dưới dạng file WAV 16-bit.
5. Kiểm tra đầu ra:
   - Đảm bảo file `output.wav` đã được tạo.
   - Đặc điểm file: WAV 16-bit, 44100 Hz, mono, khoảng 2 giây.
   - Nghe `output.wav` để đảm bảo không có biến dạng âm thanh rõ rệt so với `input.wav`.

Lưu ý: Nếu thông điệp quá dài, script sẽ cắt ngắn và in cảnh báo. Đảm bảo `alpha=0.6` đủ lớn để giải mã thành công (có thể tăng lên 0.8 nếu cần).

6 Kết quả mong đợi

Sau khi hoàn thành các tác vụ, bạn sẽ có:

- `input.wav`: File âm thanh sóng sin 2 giây, 440 Hz (mono, 16-bit, 44100 Hz).
- `preprocessed.npy`: Mảng NumPy đã chuẩn hóa của âm thanh đầu vào.
- `output.wav`: File âm thanh chứa thông điệp "A" được nhúng bằng kỹ thuật Echo Hiding Time Spread.

File `output.wav` đã sẵn sàng để giải mã trong bài thực hành tiếp theo, nơi thông điệp "A" sẽ được trích xuất với độ lỗi tối thiểu (Tỷ lệ lỗi bit $\approx$ 0%, Tương quan chuẩn hóa $\approx$ 1.0).

7 Xử lý sự cố

- **Lỗi không tìm thấy file**: Đảm bảo các file đầu vào tồn tại (`input.wav` cho Tác vụ 2, `preprocessed.npy` cho Tác vụ 3). Chạy lại các tác vụ trước nếu cần.
- **Lỗi định dạng WAV 16-bit**: Kiểm tra `input.wav` là file WAV 16-bit. Tạo lại bằng Tác vụ 1 nếu cần.
- **Âm thanh quá ngắn**: Đảm bảo `input.wav` dài ít nhất 2 giây (88200 mẫu ở 44100 Hz). Tăng `–duration` trong Tác vụ 1 nếu cần.
- **Cảnh báo clipping**: Nếu Tác vụ 3 báo clipping, đầu ra sẽ được chuẩn hóa tự động và không ảnh hưởng đến việc nhúng.
- **Vấn đề Labtainer**: Nếu container không khởi động, kiểm tra Docker đang chạy và Labtainer được cài đặt đúng. Chạy `labtainer -t` để kiểm tra thiết lập.

8 Lưu ý

- Sóng sin trong Tác vụ 1 được chọn vì tính đơn giản. Trong thực tế, âm nhạc hoặc giọng nói có thể được sử dụng, nhưng cần thêm bước tiền xử lý để giảm nhiễu.
- Các tham số `d0=200`, `d1=300`, `L=8192`, `alpha=0.6` được tối ưu cho sóng sin. Điều chỉnh trong Tác vụ 3 nếu độ chính xác giải mã thấp trong bài thực hành tiếp theo.
- Môi trường Labtainer đảm bảo tính lặp lại. Không sửa đổi phụ thuộc của container ngoài hướng dẫn.