Hướng Dẫn Thực Hành: Giải Mã 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 độ trễ khác nhau để mã hóa và giải mã 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 (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).

Để giải mã, tín hiệu được phân tích để phát hiện sự hiện diện của echo tại các độ trễ δ_0 hoặc δ_1 bằng cách tính năng lượng của các đoạn tín hiệu hoặc sử dụng các kỹ thuật tương quan.

1.2 Time Spread

Đặc trưng của kỹ thuật Time Spread là việc tổ chức tín hiệu thành các khung thời gian (frame) cố định (ví dụ: 8192 mẫu). Mỗi khung chứa một bit thông điệp, được nhúng bằng cách thêm echo tương ứng. Các khung được phân bố đều trên toàn bộ tín hiệu âm thanh, giúp tăng khả năng chống nhiễu và giảm nguy cơ phát hiện thông điệp ẩn. Để đảm bảo tín hiệu mượt mà, một cửa sổ mượt (như cửa sổ Hanning) thường được áp dụng cho các khung, giảm hiện tượng gián đoạn:

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

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:

- Giữ chất lượng âm thanh gần như nguyên vẹn, phù hợp cho ứng dụng bảo mật.
- Linh hoạt trong việc điều chỉnh tham số (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 số khung và độ dài tín hiệu.
- Yêu cầu đồng bộ chính xác giữa mã hóa và giải mã.
- Nhạy cảm với các kỹ thuật xử lý âm thanh như nén mất dữ liệu hoặc nhiễu mạnh.

Phần thực hành này tập trung vào giải mã thông điệp được nhúng bằng kỹ thuật Echo Hiding Time Spread, sử dụng các tham số đã được tối ưu để trích xuất thông điệp một cách chính xác.

2 Giới thiệu

Bài thực hành này hướng dẫn bạn cách trích xuất thông điệp ẩn trong file âm thanh bằng kỹ thuật **Echo Hiding Steganography**, dựa trên phương pháp Time Spread đã trình bày. Bài lab bao gồm ba tác vụ chính:

1. **Tác vụ 1: Chuẩn bị dữ liệu âm thanh** - Đọc và chuẩn hóa dữ liệu từ file `output.wav`.
2. **Tác vụ 2: Giải mã thông điệp** - Trích xuất thông điệp ẩn từ dữ liệu âm thanh.

3. **Tác vụ 3: Kiểm tra và xác nhận kết quả** - So sánh thông điệp đã giải mã với thông điệp gốc.

Mục tiêu là giải mã chính xác thông điệp ẩn (ví dụ: "HELLO") từ file âm thanh và đánh giá độ chính xác của kết quả. Bài lab sử dụng môi trường Labtainer để đảm bảo tính nhất quán và an toàn.

3 Yêu cầu

- **Labtainer**: Đã cài đặt trên hệ thống (Ubuntu, macOS, hoặc Windows với WSL2). Xem hướng dẫn tại: https://my.nps.edu/web/c3o/labtainers.
- **Docker**: Đảm bảo Docker được cài đặt và đang chạy.
- **Python 3 và thư viện**: Python 3, NumPy, SciPy.
- **File đầu vào**: File `output.wav` chứa thông điệp ẩn.
- Kiến thức cơ bản về Python, dòng lệnh và xử lý tín hiệu âm thanh.

4 Thiết lập bài thực hành

Bài thực hành được cấu hình trong môi trường Labtainer với các thông tin sau:

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

Để bắt đầu:

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

```
labtainer time_spread_decoding -r
```

3. Container Docker sẽ khởi động với các script và phụ thuộc cần thiết.

5 Hướng dẫn từng tác vụ

5.1 Tác vụ 1: Chuẩn bị dữ liệu âm thanh

Mục tiêu: Đọc và chuẩn hóa dữ liệu âm thanh từ file `output.wav` để chuẩn bị cho quá trình giải mã.
Script: `prepare_data.py`
Các bước:

1. Đảm bảo file `output.wav` đã có trong thư mục làm việc.
2. Chạy script để chuẩn bị dữ liệu:

```
python3 prepare_data.py
```

3. **Quá trình**:
   - Script đọc file `output.wav` và kiểm tra định dạng WAV 16-bit.
   - Dữ liệu được chuẩn hóa sang kiểu float32, biên độ trong khoảng [-1, 1].
   - Kết quả được lưu vào file `decoded_preprocessed.npy`.
4. Kiểm tra đầu ra:
   - Đảm bảo file `decoded_preprocessed.npy` đã được tạo.
   - Định dạng mong đợi: Mảng NumPy, kiểu float32, giá trị trong [-1, 1].

Lưu ý: Nếu gặp lỗi "FileNotFoundError", kiểm tra xem file `output.wav` có trong thư mục làm việc không. Nếu file không phải WAV 16-bit, script sẽ báo lỗi "ValueError".

5.2 Tác vụ 2: Giải mã thông điệp

Mục tiêu: Trích xuất thông điệp ẩn từ dữ liệu âm thanh đã chuẩn hóa.
Script: `decode_message.py`
Các bước:

1. Đảm bảo file `decoded_preprocessed.npy` từ Tác vụ 1 đã có sẵn.
2. Chạy script để giải mã:

```
python3 decode_message.py
```

3. **Quá trình**:
   - Script tải dữ liệu từ `decoded_preprocessed.npy`.
   - Phát hiện echo bằng cách tính năng lượng của từng đoạn âm thanh để xác định bit (0 hoặc 1).
   - Chuyển chuỗi bit thành văn bản ASCII.
   - Lưu thông điệp vào file `decoded_message.txt`.

4. Kiểm tra đầu ra:
    - Đảm bảo file `decoded_message.txt` đã được tạo.
    - Nội dung mong đợi: Thông điệp văn bản (ví dụ: "HELLO").

Lưu ý: Nếu thông điệp không giải mã đúng, thử điều chỉnh tham số `delay` trong script (ví dụ: tăng từ 100 lên 200 hoặc 300) để cải thiện khả năng phát hiện echo.

5.3 Tác vụ 3: Kiểm tra và xác nhận kết quả

Mục tiêu: So sánh thông điệp đã giải mã với thông điệp gốc để đánh giá độ chính xác.
Script: `verify_result.py`
Các bước:

1. Đảm bảo file `decoded_message.txt` từ Tác vụ 2 đã có sẵn.
2. Chỉnh sửa script `verify_result.py` để cập nhật thông điệp gốc (nếu cần, mặc định là "HELLO").
3. Chạy script để kiểm tra:

```
python3 verify_result.py
```

4. **Quá trình**:
    - Script đọc thông điệp gốc và thông điệp từ `decoded_message.txt`.
    - Tính tỷ lệ lỗi bit (BER) để đo tỷ lệ ký tự sai.
    - Tính tương quan chuẩn hóa (NC) để đánh giá độ tương đồng.
    - In kết quả so sánh.
5. Kiểm tra đầu ra:
    - Kết quả in ra bao gồm thông điệp gốc, thông điệp giải mã, BER và NC.
    - Kết quả lý tưởng: BER $\approx$ 0%, NC $\approx$ 1.0.

Lưu ý: Nếu BER cao hoặc NC thấp, kiểm tra lại thông điệp gốc hoặc thử điều chỉnh tham số giải mã trong Tác vụ 2.

6 Kết quả mong đợi

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

- `decoded_preprocessed.npy`: Dữ liệu âm thanh đã chuẩn hóa từ `output.wav`.
- `decoded_message.txt`: Thông điệp đã giải mã (ví dụ: "HELLO").
- Kết quả kiểm tra: Thông báo so sánh với BER $\approx$ 0% và NC $\approx$ 1.0 nếu giải mã chính xác.

7 Xử lý sự cố

- **Lỗi không tìm thấy file**: Đảm bảo `output.wav` tồn tại cho Tác vụ 1 và `decoded_preprocessed.npy` tồn tại cho Tác vụ 2. Chạy lại các tác vụ trước nếu cần.
- **Lỗi định dạng WAV**: Kiểm tra `output.wav` là file WAV 16-bit. Nếu không, sử dụng file âm thanh đúng định dạng.
- **Thông điệp giải mã không chính xác**: Điều chỉnh tham số `delay` trong `decode_message.py` (ví dụ: tăng lên 200 hoặc 300) và kiểm tra tham số mã hóa ban đầu (`d0`, `d1`, `alpha`).
- **Labtainer không khởi động**: Kiểm tra Docker đang chạy và Labtainer được cài đặt đúng. Chạy `labtainer -t` để kiểm tra.

8 Lưu ý

- File `output.wav` phải được tạo từ bài lab mã hóa trước đó với các tham số phù hợp (`d0=200`, `d1=300`, `alpha=0.6`, `L=8192`).
- Tham số `delay=100` trong `decode_message.py` là giá trị mặc định. Cần điều chỉnh để khớp với `d0` và `d1` của quá trình mã hóa.
- Môi trường Labtainer đảm bảo tính lặp lại. Không sửa đổi phụ thuộc của container ngoài hướng dẫn.

9 Nộp bài

- **File cần nộp**:
  - `decoded_preprocessed.npy`: Dữ liệu đã chuẩn hóa.
  - `decoded_message.txt`: Thông điệp đã giải mã.
  - Ảnh chụp màn hình kết quả chạy `verify_result.py` (hiển thị BER và NC).
- **Hình thức nộp**: Nén các file vào một file `.zip` và nộp qua hệ thống được chỉ định.
- **Đánh giá**: Dựa trên độ chính xác của thông điệp giải mã (BER $\approx$ 0%, NC $\approx$ 1.0) và đầy đủ các file yêu cầu.