Politechnika Warszawska

Dokumentacja Projektu

Metody wykrywania steganografii

Przedmiot: Kryptografia Stosowana
Kod przedmiotu: 103A-TLTIC-MSP-KRYS
Prowadzący: dr Adam Komorowski

Autorzy:

Safiya Aliaksandrava
Wiktor Ciołek, 311501
Ignacy Czajkowski
Urszula Kamińska
Dawid Karpiński, 311383

Data:
9 stycznia 2025

1 Wstęp

Czym jest staganografia
kto jest autorem czego
Link do repozytorium z kodami

2 Wykrywanie LSB Matching w obrazach za pomocą uczenia maszynowego

2.1 Opis steganografii

Jedną z najpopularniejszych metod steganografii obrazów jest *LSB*[5]. Każda z jej odmian opiera się na modyfikacji najmniej znaczącego bitu informacji opisującej kolor pikseli. Autor ukrywanej w ten sposób wiadomości musi wybrać piksele (w przypadku kodowania RGB dodatkowo kanał), które zmodyfikuje. Odbiorca tej wiadomości może ją odczytać na podstawie zmienionego obrazu oraz reguły opisującej kolejność odczytywania bitów.

LSB zawdzięcza swoją popularność co najmniej dwóm faktom. Po pierwsze jest to prostolinijna idea, łatwa do opisania. Jeszcze prostsze jest odczytanie zakodowanej wiadomości. Należy tu jednak zaznaczyć, że nie należy mylić prostolinijności idei z trywialnością implementacji. Otóż sama implementacja nie musi być trywialna, o czym napiszemy w następnym akapicie dotyczącym różnych wersji *LSB*. Druga ważna cecha tej steganografii to niemalże niemożliwe jej wykrycie za pomocą inspekcji wizualnej. Zawdzięczane jest to temu, że zmieniane są tylko najmniej znaczące bity, które mają najmniejszy wpływ na kolor widoczny na obrazie.

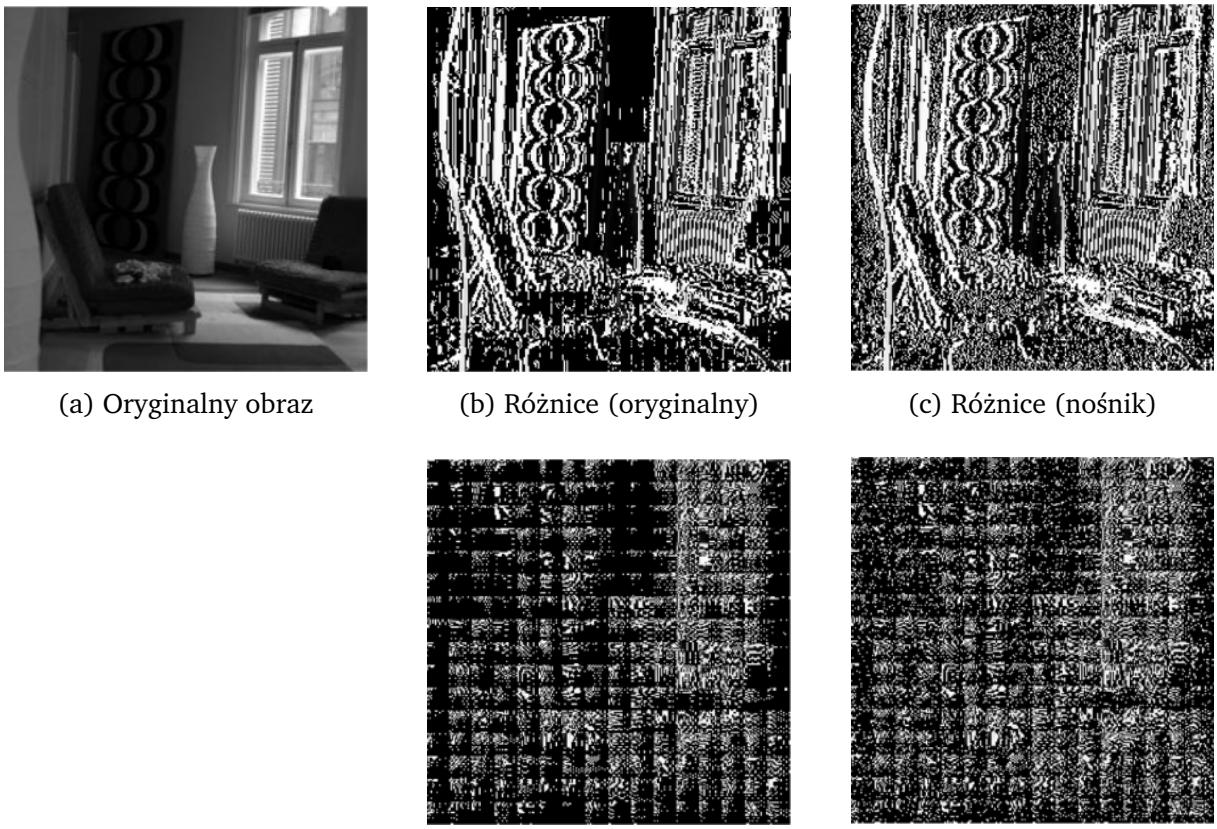
Przejdziemy teraz do omówienia szczególnej wersji *LSB* jaką jest *LSB Matching*. Zakładamy, że mamy wybrany bajt G_i, w którym chcemy zapisać część wiadomości, czyli bit m_i. Jeśli najmniej znaczący bit bajtu jest równy m_i to pozostawiamy go niezmienionego. W przeciwnym przypadku z prawdopodobieństwem $p = 0.5$ dodajemy lub odejmujemy 1 od bajtu G_i. Wyjątkami są przypadki jeśli nowa wartość bajtu nie byłaby dozwolona w danym kodowaniu. Wtedy wybierana jest dozwolona opcja. Np. w kodowaniu 8-bitowym odcieniu szarości (*8-bit grayscale*) musi zachodzić $0 <= G_i <= 255$. [9]

Takie podejście jest adaptacją innej wersji zwanej *LSB Subsitution*, w której najmniej znaczący jest po prostu zamieniany na pożądaną wartość. Taka steganografia jest jednak bardzo podatna na atak statystyczny, który polega na zbadaniu histogramu występujących wartości bajtów i np. wykonaniu testu χ^2. Metoda *LSB Matching* jest odporna na takie ataki i sprawia kłopoty analizom statystycznym przez dywersyfikacje wprowadzanych zmian.[6, 9]

Za pomocą metody *LSB* można przekazać wiadomość o rozmiarze co najwyżej takim, ile bajtów koduje dany obraz. Przykładowo, jeśli jest to obraz trójkolorowy o rozmiarze 256x256 pikseli, można za jego pomocą ukryć wiadomość o długości $3 \cdot 256 \cdot 256 = 192kb$. Jednak im więcej ukrywający wiadomość chce wykorzystać tym bardziej narażony jest na atak. To jaka część obrazu jest wykorzystywana do ukrywania będziemy nazywali jako *usage* $\mu \in [0, 1]$

2.2 Opis metody wykrywania

To że, *LSB Matching* jest odporne na ataki czysto statystyczne, nie oznacza, że nie pozostawia żadnych śladów. Poniższe rysunki pokazują jakie zmiany są obserwowalne pomiędzy czystymi a naruszonymi obrazami (patrz rysunek 1).



(a) Oryginalny obraz (b) Różnice (oryginalny) (c) Różnice (nośnik)

(d) DCT (oryginalny) (e) DCT (nośnik)

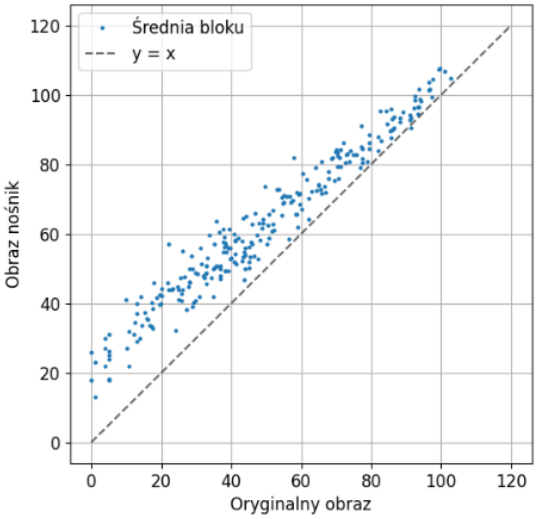
Rysunek 1: Porównanie różnic pomiędzy sąsiadującymi pikselami w poziomie oraz DCT dla obrazu oryginalnego i z zakodowaną wiadomością

Pierwsza przedstawiona transformacja to różnica pomiędzy sąsiadującymi w poziomie pikselami. Na oryginalnych obrazach występują obszary, gdzie kolor jest taki sam, przez co nie występują różnice między sąsiadującymi pikselami. Na naruszonych obrazach w tych samych obszarach występują różnice. Drugą przedstawioną transformacją jest *DCT* (*Discrete Cosine Transform*) opisana wzorem 1. Obraz dzielony jest na bloki o wymiarach NxN (na rysunku i dalej $N = 16$), z których każdy jest poddawany przekształceniu. Interpretacja współczynników DCT jest następująca. Współczynnik $X_{k,\ell}$ odpowiada tym wyższej częstotliwości pionowej (zmianom wartości bajtów w poziomie) im większe k oraz tym większej częstotliwości poziomej im większe ℓ.

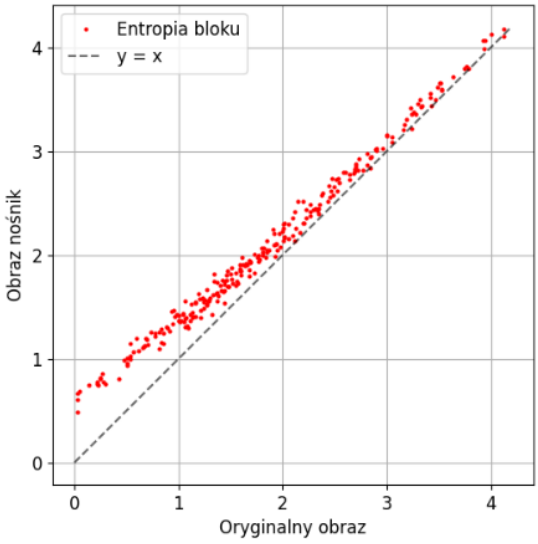
$$X_{k,\ell} = \sum_{m=0}^{N-1} \sum_{n=0}^{N-1} x_{m,n} \cos\left[\frac{\pi}{N}\left(m+\frac{1}{2}\right)k\right] \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)\ell\right] \tag{1}$$

Kluczową obserwacją, którą wykorzystamy, jest to, że ukrywanie wiadomości w obrazie zwiększa obecność wysokich częstotliwości przestrzennych, szczególnie w obszarach o oryginalnie małych zmianach koloru.

Poza zmianami w różnicach pomiędzy sąsiadującymi pikselami oraz wysokoczęstotliwościowych składowych *DCT* wskazać należy jeszcze jedną charakterystyczną, mierzalną zmianę wprowadzaną przez *LSB Matching*. Okazuje się, bowiem, że im mniejsza jest średnia oraz entropia Shannona w blokach *DCT*, tym bardziej większa jest ona w odpowiadających blokach z naruszonych obrazów. Poniższy rysunek (patrz rysunek 4) przedstawia porównanie pomiędzy tymi samymi blokami *DCT* wziętymi z oryginalnego obrazu i nośnika. Obserwowalne jest zwiększenie się średniej i entropii Shannona. Choć takie założenie niemalże nigdy nie jest spełnione, warto wspomnieć ten fakt, że gdyby atakujący tę steganografię miał dostęp do obu obrazów i mógł je ze sobą porównać, w większości przypadków na podstawie samych tych wielkości mógłby wskazać ten z zakodowaną wiadomością.



Rysunek 2: Średnia



Rysunek 3: Entropia

Rysunek 4: Zmiany w średniej i entropii Shannona bloków DCT. Porównanie obrazu oryginalnego z nośnikiem. Każdy punkt reprezentuje odpowiadający blok o wymiarach 16x16

Najwyższa pora, aby zacząć mówić o samej metodzie wykrywania *LSB Matching*. Jak pisze na swoim blogu Daniel Lerch, jedyną możliwością wykrycia tej stenografii na podstawie samego podejrzanego obrazu są metody uczenia maszynowego [6]. Omówione powyżej cechy mogą zostać wykorzystane jako dane wejściowe do modelu klasyfikującego obraz jako nośnik bądź czysty. O tym jaki model został wybrany oraz na jaki dokładnie wektor cech został użyty w implementacji powiemy w następnej sekcji.

Zanim przejdziemy dalej warto wspomnieć o ograniczeniach i specyfikacji tej metody. Przede wszystkim opisywana metodologia opiera się na założeniu, że znana jest steganografia, czyli, że jest to *LSB Matching*. Nie można zatem zastosować wytrenowanego modelu do wykrywania innej steganografii. Ponadto, istnieją inne typy opierające się na modyfikacji najmniej

znaczącego bitu, które mogą implementować sposób wyboru pikseli, który jest mniej naiwny niż wybór losowy, klasyczny [5]. Do takich należy *LSB Matching Revisited* opisany przez Mielikainena w 2006 oraz jego warianty [7, 3]. Model został wytrenowany przy użyciu obrazów z wiadomościami zakodowanymi w sposób zupełnie losowy i może nie być skuteczny w wykrywaniu obrazów z wiadomością zakodowaną w pikselach wybranych na podstawie wiedzy o obrazie. Kolejnym ograniczeniem metody uczenia maszynowego jest potrzeba przygotowania zbioru obrazów zarówno czystych jak i z ukrytą wiadomością. Dodatkowo metody uczenia maszynowego często mogą wymagać większej mocy obliczeniowej, ze względu na potrzebę obróbki obrazów i ekstrakcji wyżej opisanych cech.

2.3 Implementacja

W ramach implementacji metody wykrywania *klasycznego LSB Matching* wykonano następujące kroki

1. **Przygotowanie danych i implementacja steganografii**
   Skorzystano ze zbioru `BOSSbase` zawierającego 10 tysięcy obrazów o wymiarach 256x256 pikseli każdy o wielkości bajta [1]. W każdym z obrazów ukryto dodatkowo wiadomość z parametrem $\mu = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1$. Otrzymano w ten sposób dla każdego obrazu dodatkowo różne wersję każda z ukrytą wiadomością o innej długości. Wiadomością był losowy ciąg bitów. Ukrywany był on w losowo wybieranych pikselach

2. **Przygotowanie funckji do ekstrakcji cech**
   Jako wektor cech do uczenia modelu wybrano:

   - średnia wartość współczynników z każdego z $256 \cdot 256/(N = 16)^2 = 256$) bloków *DCT*
   - entropia współczynników z każdego z bloków *DCT*
   - 1 współczynnik bloku *DCT* odpowiadający najwyższym częstotliwościom przestrzennym

   Łącznie $3 \cdot 256 = 768$ elementów wektora cech dla danego obrazu.

3. **Wybór algorytmu i uczenie modelu**
   Jako model do klasyfikacji obrazu wybrano *las losowy*. *Las losowy* jest *ensemblem* składającym się z drzew decyzyjnych konstruowanych na podstawie losowo wybieranych cech. Ostatecznie, klasyfikacja polega na demokratycznym wyborze większościowym spośród wszystkich estymatorów. W implementacji użyto n=100 drzew decyzyjnych.[2]
   Do uczenia użyto po 100 obrazów czystych i nośników oraz z każdego przygotowanego wykorzystania μ, czyli łącznie 2000 obrazów

Kod wykorzystany do implementacji znajduje się w repozytorium w folderze *\image_lsbm_ml*. Wymagania środowiska Python znajdują się w pliku *requirments.txt*

2.4 Wyniki, interpretacja i wnioski

Model został przetestowany na zbiorze testowym oraz dla różnych stopni wykorzystania pikseli μ. Utworzony w ten sposób wykres przedstawiony jest na rysunku 5. Czułość i swoistość są obliczane według odpowiednio wzorów 2 i 3, gdzie TP - poprawnie wskazane steganografie, FP - fałszywie wskazane steganografie, FN - fałszywie odrzucone steganografie, TN - poprawnie odrzucone steganografie.

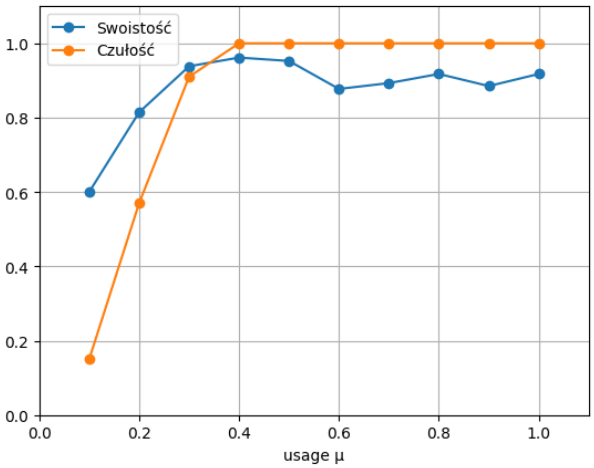
$$C = \frac{TP}{TP + FN} \tag{2}$$

$$S = \frac{TP}{TP + FP} \tag{3}$$

Im wyższa czułość tym rzadziej model niezauważa steganografii. Jest to cecha pożądana ze względu na potrzebę wskazania obrazów, które mogą potencjalnie przenosić informacje. Zauważamy, że im więcej pikseli zostało wykorzystanych do przekazania wiadomości, tym lepiej model radzi sobie ze wskazywaniem ich. Przy wykorzystaniu pikseli μ większym niż około 40% każda steganografia jest zauważona. Model nie radzi sobie z mniej zmodyfikowanymi obrazami co jest zgodne z intuicją.

Im wyższa z kolei swoistość tym rzadziej model wszczyna fałszywy alarm. Jest to znów pożądana cecha szczególnie, gdy chcemy wszcząć zaawansowane procedury przy wykryciu steganografii. Ta wielkość rośnie dla małych μ a następnie oscyluje w okolicy 0.9.

Wyniki modelu można ocenić pozytywnie ze względu na wysokie wartości zarówno swoistości jak i czułości. Słaba skuteczność w przypadku małych μ nie jest zatrważający szczególnie jeśli weźmie się pod uwagę, że przy tak małej dostępnej długości wiadomości niewiele da się przekazać. Jednocześnie warto przyznać, że jest to niedoskonałość i dalsze badania udoskonalające proces uczenia i model mogłyby postawić za cel zmienienie zniwelowanie tego efektu.



Rysunek 5: Czułość i swoistość wytrenowanego modelu *lasu losowego* w funkcji wykorzystania pikseli μ

3 Wykrywanie Echo Hiding poprzez analizę cepstrum mocy sygnału

3.1 Echo Hiding

Jednym z najczęstszych podejść w dziedzinie steganografii audio jest metoda zwana *Echo Hiding*, *EH*. Metoda ta wykorzystuje właściwości dźwięku, dzięki którym dodatkowe echo można wprowadzić w sposób niesłyszalny dla ludzkiego ucha, jednocześnie kodując w nim wiadomość [4].

Podstawowa idea EH polega więc na dodaniu niewielkiego, kontrolowanego opóźnienia do oryginalnego sygnału audio. W celu zakodowania wiadomości, sygnał audio $x(n)$ poddawany jest operacji splotu z funkcją impulsową $h(n)$, która definiuje właściwości echa. Matematycznie proces ten można zapisać jako:

$$y(n) = h(n) * x(n), \tag{4}$$

gdzie operator $*$ oznacza splot, a $h(n)$ jest funkcją kernela zdefiniowaną jako:

$$h(n) = \delta(n) + a\delta(n-d). \tag{5}$$

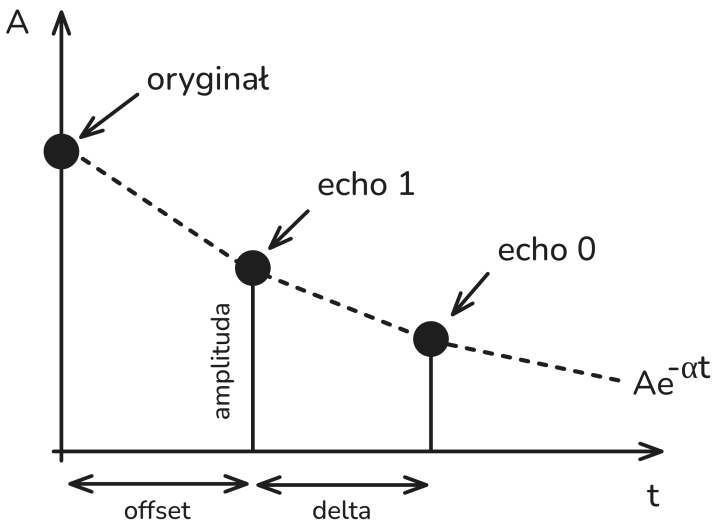
W powyższym równaniu $\delta(n)$ oznacza deltę Diraca, a jest współczynnikiem tłumienia echa ($0 < a < 1$), a d to opóźnienie echa wyrażone w ilości próbek.

Wartości d oraz a są zwykle dobierane w taki sposób, aby echo pozostało niesłyszalne, jednocześnie zapewniając wystarczające właściwości by umożliwić dekodowanie ukrytej wiadomości.

Przed kodowaniem, wejściowy sygnał audio dzieli się na segmenty o stałej długości N, w taki sposób, by każdy zawierał echo reprezentujące jeden bit wiadomości. Zamienia się więc ją na reprezentację binarną, a następnie koduje poprzez użycie dwóch różnych opóźnień: d_0, dla bitu "0", oraz d_1, dla bitu o wartości "1".

Zatem, osadzane dane są definiowane za pomocą czterech głównych parametrów echa (6):

1. amplituda echa (A),
2. współczynnik zanikania (α),
3. opóźnienie dla bitu "1" (*offset*),
4. opóźnienie dla bitu "0" ($offset + delta$).



Rysunek 6: Parametry EH.

3.2 Dekodowanie

Opisany poniżej proces dekodowania został wykonany na przykładzie ukrytego echo o parametrach: $d_1 = 48$ próbek, $d_0 = 96$ próbek, i długości segmentu $N = 1078$ próbek, w pliku audio z nagraną ludzką mową.

3.2.1 Analiza cepstralna i cepstrum mocy

Najczęstszą procedurą wykorzystywaną do wykrywania echa jest analiza cepstralna. Obliczenie tzw. cepstrum $C_y(n)$ pozwala zidentyfikować opóźnienia dodanego echa, ponieważ zawiera wyraźne szczyty w odpowiadających im pozycjach.

Cepstrum $C_y(n)$ definiowane jest jako odwrotna transformata Fouriera logarytmu widma amplitudowego sygnału:

$$C_y(n) = \mathscr{F}^{-1}\big(\ln|\mathscr{F}(y(n))|\big), \tag{6}$$

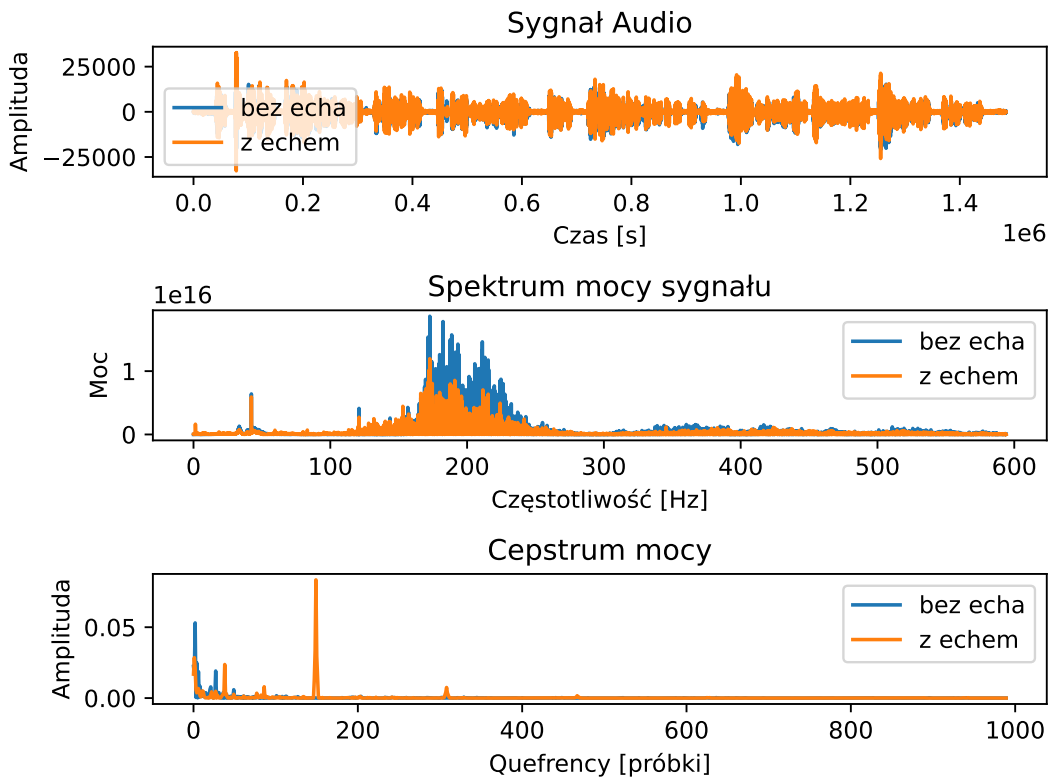
gdzie $\mathscr{F}$ oznacza transformatę Fouriera, $y(n)$ to sygnał z ukrytym echem, a $\mathscr{F}^{-1}$ oznacza odwrotną transformatę Fouriera.

Można również zdefiniować tzw. cepstrum mocy (ang. *power cepstrum*), które daje lepsze wyniki niż samo cepstrum [8]. Oblicza się je jako odwrotną transformatę Fouriera logarytmu mocy widma sygnału:

$$P_y(n) = \left|\mathscr{F}^{-1}\big(\ln|\mathscr{F}(y(n))|^2\big)\right|^2, \tag{7}$$

```
def power_cepstrum(audio):
    spectrum = np.abs(np.fft.fft(audio)) ** 2
    log_spectrum = np.log(spectrum + np.finfo(float).eps)
    power_cepstrum = np.abs(np.fft.ifft(log_spectrum)) ** 2
    return power_cepstrum
```

Listing 1: Fragment kodu liczącego cepstrum mocy.

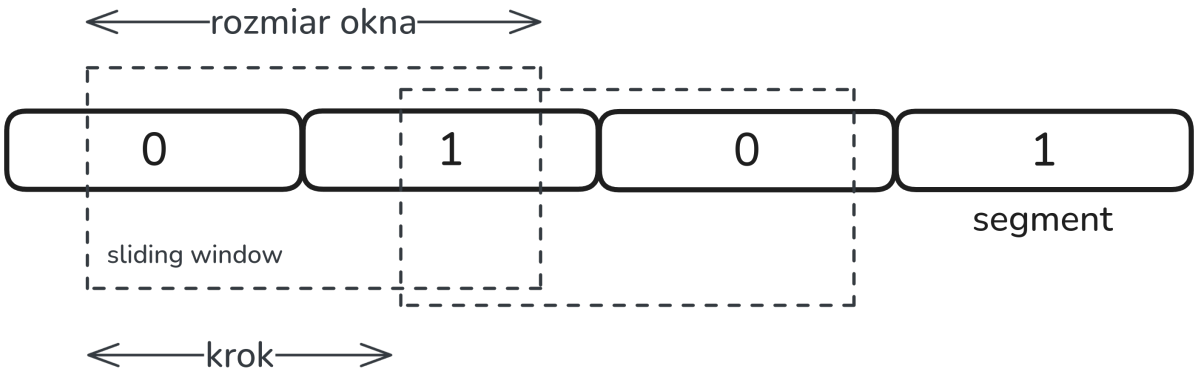


Rysunek 7: Porównanie reprezentacji sygnału z echem i bez echa.

3.2.2 Estymacja opóźnień echa

Dokonując steganalizy jako potencjalny atakujący, nie mamy pojęcia o użytych parametrach do kodowania echem. Natomiast, możliwa jest ich estymacja poprzez wykorzystanie analizy cepstrum mocy z przesuwanym oknem (ang. *sliding window*) [10].

W tej metodzie sygnał analizowany jest segment po segmencie za pomocą okna o określonym rozmiarze τ, które przesuwa się po sygnale z krokiem k (8). Dla każdego segmentu obliczane jest cepstrum mocy, a następnie lokalizowane są szczyty odpowiadające potencjalnym opóźnieniom d_0 i d_1. Po przesunięciu okna przez cały sygnał tworzy się histogram lokalizacji pików w celu identyfikacji najbardziej prawdopodobnych wartości opóźnień.



Rysunek 8: Wizualizacja algorytmu okna przesuwnego.

Algorytm został zaimplementowany w klasie `EchoDetector`, której wejściowe parametry to:

- `window_size` – długość okna analizy w próbkach,
- `step_size` – krok przesuwania okna w próbkach,
- `min_delay` – minimalny czas opóźnienia, który będzie analizowany,
- `max_delay` – maksymalny czas opóźnienia, który będzie analizowany.

Cepstrum mocy jest obliczane dla kolejnych segmentów sygnału, wyznaczanych za pomocą okna przesuwnego, z dodatkowo aplikowaną funkcją Hamminga. Metoda `iter_cepstrums` generuje cepstrum dla kolejnych okien:

```
def iter_cepstrums(self, audio, *, window_size=None, step_size=None):
    window_size = window_size or self.window_size
    step_size = step_size or self.step_size
    for idx in range(0, (audio.size - window_size) + 1, step_size):
        window = audio[idx : idx + window_size]
        window = window * np.hamming(window_size)
        yield power_cepstrum(window)
```

Listing 2: Generator liczący cepstrum mocy dla poszczególnych okien przesuwnych.

Na podstawie obliczonego cepstrum, metoda `estimate_delays` wyznacza histogram lokalizacji maksimów w przedziale `min_delay`–`max_delay`. Dwa najczęściej występujące maksima (`d0` i `d1`) są uznawane za potencjalne opóźnienia echa:

```
counts, bins = np.histogram(
    peak_locations,
    bins=(self.max_delay - self.min_delay),
    range=(self.min_delay, self.max_delay),
)
```

```
top_2 = np.argsort(counts)[-2:]
d0, d1 = bins[top_2]
```

Listing 3: Tworzenie histogramu z lokalizacji pików.

3.2.3 Współczynnik CPLAR

Jedną z kluczowych wielkości oceniających jakość detekcji opóźnień jest współczynnik **CPLAR** (*Cepstrum Peak Location Aggregation Rate*) [10]. Jest definiowany jako stosunek liczby okien cepstrum, w których piki zostały poprawnie zlokalizowane w pozycjach d_0 lub d_1, do całkowitej liczby analizowanych okien.

```
cplar = sum(counts[top_2]) / len(peak_locations)
```

Listing 4: Obliczanie współczynnika CPLAR.

3.2.4 Estymacja długości segmentu

Po otrzymaniu opóźnień echa można wyznaczyć długość segmentu w przybliżony sposób, poprzez iterowanie po potencjalnych długościach i wyznaczanie na podstawie każdego segmentu średnią sumę wartości cepstrum w lokalizacjach `d0` i `d1` [10]:

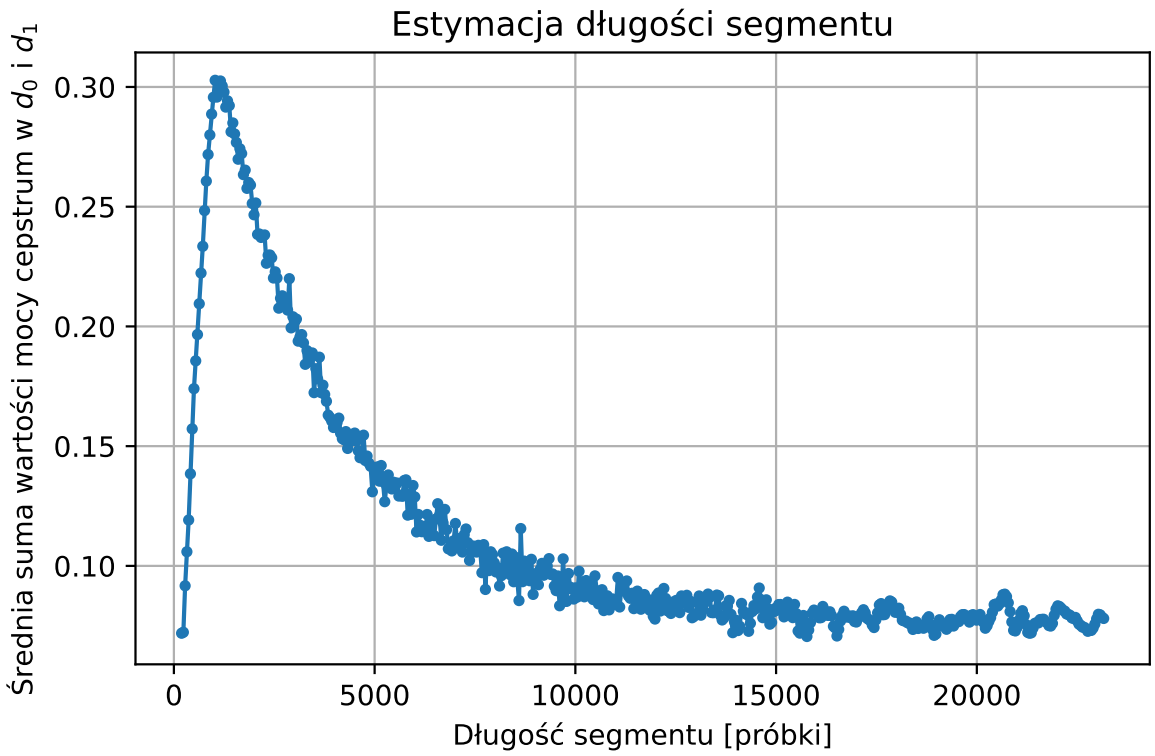
```
avg_sum = np.mean(
    [
        cepstrum[d0] + cepstrum[d1]
        for cepstrum in self.iter_cepstrums(
            audio,
            window_size=est_length,
            step_size=est_length,
        )
    ]
)
```

Listing 5: Średnia z sum wartości pików z każdego segmentu.

Ostateczna, estymowana długość segmentu to ta, dla której średnia suma wartości mocy w lokalizacjach `d0` i `d1` jest największa (9):

```
return lengths[np.argmax(avg_sums)]
```

Listing 6: Branie długości, której odpowiada średnia suma o maksymalnej wartości



Rysunek 9: Wykres średniej sumy pików od badanej, potencjalnej długości segmentu.

Na koniec, znając już wszystkie potrzebne parametry można obliczyć długości segmentów N, z jakimi zakodowana jest wiadomość.

```
num_bits = result.size // segment_len
bits = np.zeros(num_bits).astype(np.uint8)

for idx, cepstrum in zip(
    range(num_bits),
    detector.iter_cepstrums(
        result,
        window_size=segment_len,
        step_size=segment_len,
    ),
):
    peak_one = d1 < cepstrum.size and cepstrum[d1] or 0
    peak_zero = d0 < cepstrum.size and cepstrum[d0] or 0
    bits[idx] = int(peak_one < peak_zero)
```

Listing 7: Ostateczna procedura dekodowania wiadomości.

3.3 Podsumowanie

W ramach projektu zaimplementowano algorytmy pozwalające na detekcję opóźnień d_0 i d_1, oraz oszacowanie długości segmentów N. Kluczowe równania, takie jak obliczanie cepstrum mocy oraz proces analizy z przesuwanym oknem, zostały zaimplementowane w środowisku Python.

4 Podsumowanie

Źródła

[1] *BOSSbase*. URL: https://www.kaggle.com/datasets/lijiyu/bossbase. (dostęp: 9 stycznia 2025).

[2] Leo Breiman. "Random Forests". In: *Machine Learning* 45.1 (Oct. 2001), pp. 5–32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324. URL: https://doi.org/10.1023/A:1010933404324.

[3] Mansoor Fateh, Mohsen Rezvani, and Yasser Irani. "A New Method of Coding for Steganography Based on LSB Matching Revisited". In: *Security and Communication Networks* 2021.1 (2021), p. 6610678. DOI: https://doi.org/10.1155/2021/6610678. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1155/2021/6610678. URL: https://onlinelibrary.wiley.com/doi/abs/10.1155/2021/6610678.

[4] Daniel F. Gruhl, Anthony Lu, and Walter Bender. "Echo Hiding". In: *Information Hiding*. 1996. URL: https://api.semanticscholar.org/CorpusID:14156378.

[5] Meike Helena Kombrink, Zeno Jean Marius Hubert Geradts, and Marcel Worring. "Image Steganography Approaches and Their Detection Strategies: A Survey". In: *ACM Comput. Surv.* 57.2 (Oct. 2024). ISSN: 0360-0300. DOI: 10.1145/3694965. URL: https://doi.org/10.1145/3694965.

[6] Daniel Lerch. *LSB Matching and Matrix Embedding*. URL: https://daniellerch.me/image-stego-lsbm/. (dostęp: 9 stycznia 2025).

[7] J. Mielikainen. "LSB matching revisited". In: *IEEE Signal Processing Letters* 13.5 (2006), pp. 285–287. DOI: 10.1109/LSP.2006.870357.

[8] Chen Ning and Zhu Jie. "A Novel Echo Detection Scheme based on Autocorrelation of Power Cepstrum". In: *2007 Second International Conference on Communications and Networking in China*. 2007, pp. 554–558. DOI: 10.1109/CHINACOM.2007.4469451.

[9] Tanmoy Sarkar and Sugata Sanyal. *Steganalysis: Detecting LSB Steganographic Techniques*. 2014. arXiv: 1405.5119 [cs.MM]. URL: https://arxiv.org/abs/1405.5119.

[10] Chunhui Xie, Yimin Cheng, and Yangkun Chen. “An active steganalysis approach for echo hiding based on Sliding Windowed Cepstrum”. In: *Signal Process.* 91.4 (Apr. 2011), pp. 877–889. ISSN: 0165-1684. DOI: 10.1016/j.sigpro.2010.09.006. URL: https://doi.org/10.1016/j.sigpro.2010.09.006.