Niezawodność i diagnostyka układów cyfrowych

Skład grupy:

- Kewin Warzecha, 249451, Śr TN 7:30
- Artur Sołtys, 248854, Śr TP 7:30

Cele projektu:

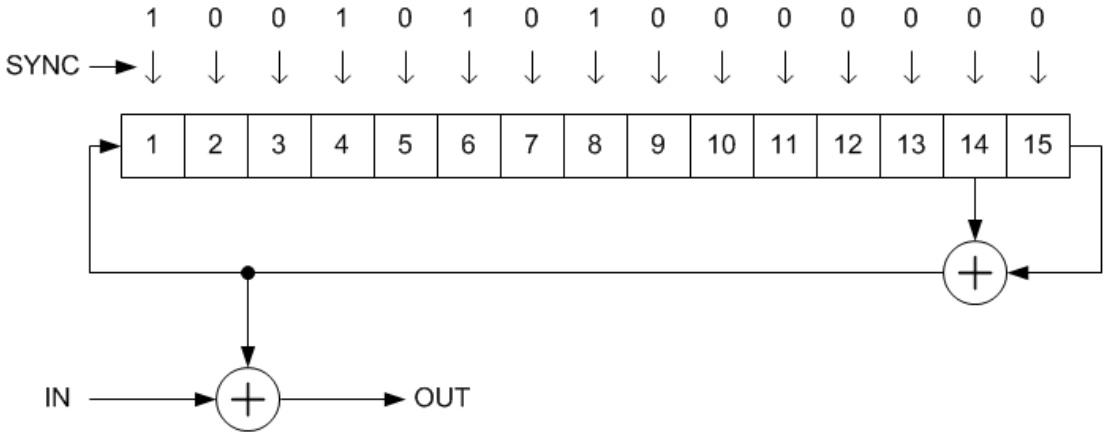
- Badanie długości ciągów binarnych przy zastosowaniu różnych algorytmów scramblowania.
- Symulacja przesyłu danych zrandomizowanych różnymi algorytmami, oraz bez randomizacji dla różnych rodzajów danych wejściowych (ciągi pseudolosowe, pliki – np. bitmapy).
- Analiza i dobór optymalnych parametrów systemu dla założonej charakterystyki zakłóceń oraz minimalnej szybkości przesyłania w sposób minimalizujący współczynnik Bit Error Rate. Wyznaczenie parametrów opisujących efektywność transmisji.

Założenia do realizacji:

Zakładamy, że istnieją ciągi bitowe bardziej prawdopodobne niż inne, lecz trudniejsze do transmisji. Scrambler – koder - randomizuje ciągi na łatwiejsze do przesyłania. Powstają one w wyniku sumowania za pomocą operacji XOR danych z pseudolosowymi wartościami, w wyniku czego powstaje maksymalnie długa sekwencja, przesyłana następnie torem transmisyjnym. Descrambler dekoduje informacje do postaci pierwotnej. W obydwu urządzeniach używa się rejestrów przesuwnych.

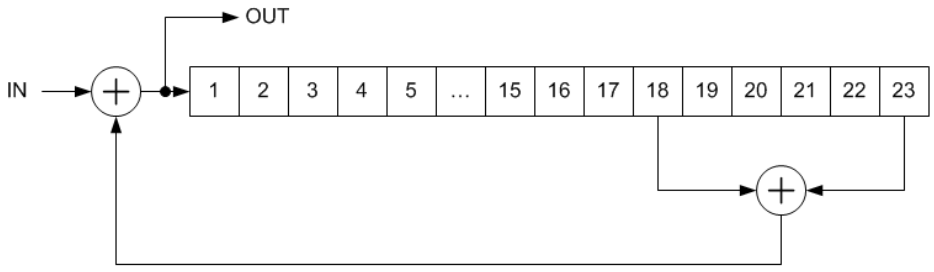
Do realizacji projektu i analizy statystycznej zostanie wykorzystane środowisko Matlab. Sprawdzane będzie działanie dwóch rodzajów scramblerów: addytywnego oraz multiplikatywnego, a następnie zostanie wykonana ocena i porównanie ich działania.

Rysunek 1: Scrambler addytywny



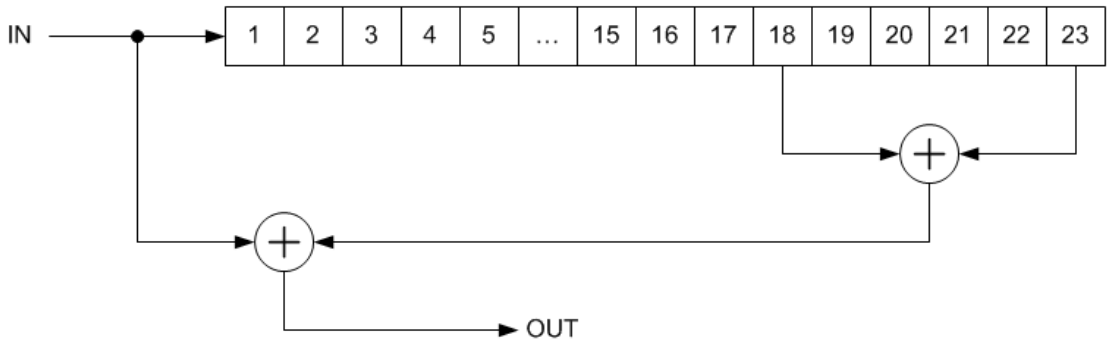
Scrambler addytywny - przekształcają strumień danych wejściowych poprzez zastosowanie pseudolosowej sekwencji binarnej. Często jest on realizowany przez rejestr przesuwny z liniowym sprzężeniem zwrotnym (LFSR). Aby zapewnić synchroniczną pracę nadawczej i odbierającej LFSR (kodera i dekodera), należy użyć słowa synchronizacji. Słowo synchronizacji jest wzorcem umieszczanym w strumieniu danych w równych odstępach czasu (w każdej ramce). Odbiornik wyszukuje kilka słów synchronizacji w sąsiednich ramkach, a zatem określa miejsce, w którym jego LFSR musi zostać ponownie załadowany ze wstępnie zdefiniowanym stanie początkowym. Addytywny descrambler jest tym samym co addytywny scrambler. Jest on zdefiniowany przez wielomian jego LFSR, dla powyższego obrazka jest to $(1+z^{-14}+z^{-15})$ i jego stan początkowy.

Rysunek 2: Scrambler multiplikatywny



Multiplikatywne scramblery (znane również jako przepust) są tak nazywane, ponieważ pełnią one mnożenia sygnału wejściowego przez koder w funkcji przenoszenia , w Z przestrzeni . Są to dyskretne liniowe systemy niezmienne w czasie . Multiplikatywny scrambler jest rekurencyjny, a multiplikatywny deszyfrator nie jest rekurencyjny. W przeciwieństwie do scramblerów addytywnych, scramblery multiplikacyjne nie potrzebują synchronizacji ramek, dlatego nazywane są również samosynchronizującymi . Multiplikatywny szyfrator / deszyfrator jest definiowany podobnie przez wielomian (dla szyfratora na zdjęciu jest to $1+z^{-18}+z^{-23}$), który jest również funkcją przesyłania descramblera.

Rysunek 3: Descrambler multiplikatywny



Uruchomienie

Aby uruchomić projekt należy mieć zainstalowane środowisko MATLAB. Następnie należy zaimportować folder z plikami z repozytorium. Można to zrobić za pomocą polecenia:

>git clone https://github.com/kekusss/NiDUC.git

Aby uruchomić symulator wystarczy wpisać w lini komend programu MATLAB:

>program

Dane wejściowe

Program domyślnie wykonuje testy na czterech rodzajach sygnalow wejściowych :

- sygnał pseudolosowy ze zwiększoną ilością ciągów binarnych
- sygnał samych jedynek
- sygnał samych zer
- sygnal którego źródłem jest plik, na przykład w formacie .jpg

Konfiguracja

Zmiana długości ciągów generowanych w programie jest możliwa za pomocą zmiennej chainLength znajdującej się w program.m, domyślnie przyjmuje ona wartość 10000.

```
7
8 -  %% dlugosc ciagow wejsciowych sygnalow
9
10
11 - chainLength = 10000;
12
%% Przygotowanie sygnalow do przeprowadzenia testow
[randomSignal,onesSignal,zerosSignal,fileSignal] = generateTestSignals(chainLength);
```

W przypadku pobierania danych z pliku jego nazwę możemy zmienić w pliku generateTestSignals.m w funkcji fopen(filename), domyślnie jest to file.jpg:

```
17      %% Wczytywanie z pliku
18 -    fileID = fopen('file.jpg');
19 -    signal = fread(fileID);
20 -    for i=1:1:length(signal)
21 -        signal(i,1)=mod(signal(i,1),2);
22 -    end
23 -    fileSignal=signal';
```

Porównanie ilości i długości ciągów binarnych jest możliwe za pomocą funkcji:

- DVBTest(sygnał, wyrownanie)
- V34Test(sygnał, wyrownanie)

gdzie w miejsce parametru sygnał przekazujemy sygnał który chcemy poddać randomizacji i testom, a w miejsce parametru wyrownanie wstawiamy 0 (tylko dla ciagow zer i jedynek) lub 1 (w pozostałych przypadkach).

```
%% Testowanie scramblera addytywnego DVB dla sygnalu losowego, samych zer, samych jedynek
DVBTest(randomSignal, 1);
DVBTest(onesSignal, 0);
DVBTest(zerosSignal, 0);
DVBTest(fileSignal, 1);

%% Testowanie scramblera multiplikatywnego V34 dla sygnalu losowego, samych zer, samych jedynek
V34Test(randomSignal, 1);
V34Test(onesSignal, 0);
V34Test(zerosSignal, 0);
V34Test(fileSignal, 1);
```

Możliwe jest również wyznaczenie współczynnika **Bit Error Rate** za pomocą funkcji BERTests(sygnał1, sygnał2, sygnał3, sygnał4, czy randomizacja), która porównuje 4 sygnały wejściowe i wartość 0 lub 1, która informuje czy poddajemy testowany sygnał scramblowaniu.

```
%% Przeprowadzenie testow dla roznych sygnalow wyznaczajac wspolczynnik BER
BERTests(randomSignal, zerosSignal, onesSignal, fileSignal, 0); % test dla sygnalu bez randomizacji
BERTests(randomSignal, zerosSignal, onesSignal, fileSignal, 1); % test dla sygnalu z randomizacja
```

Przebieg serii badań:

W celu porównania scramblerów i dokładniejszego zbadania ich właściwości wykonaliśmy serię testów dla różnych typów ciągów binarnych, między innymi dla:

- ciągu pseudolosowego ze zwiększoną częstotliwością występowania ciągów tych samych bitów
- ciągu składającego się z samych jedynek
- ciągu składającego się z samych zer
- ciągu binarnego będącego wynikiem wczytania zawartości pliku jpg

Badania wykonaliśmy także dla różnych długości ciągów wejściowych:

- 10
- 100
- 1000
- 10000

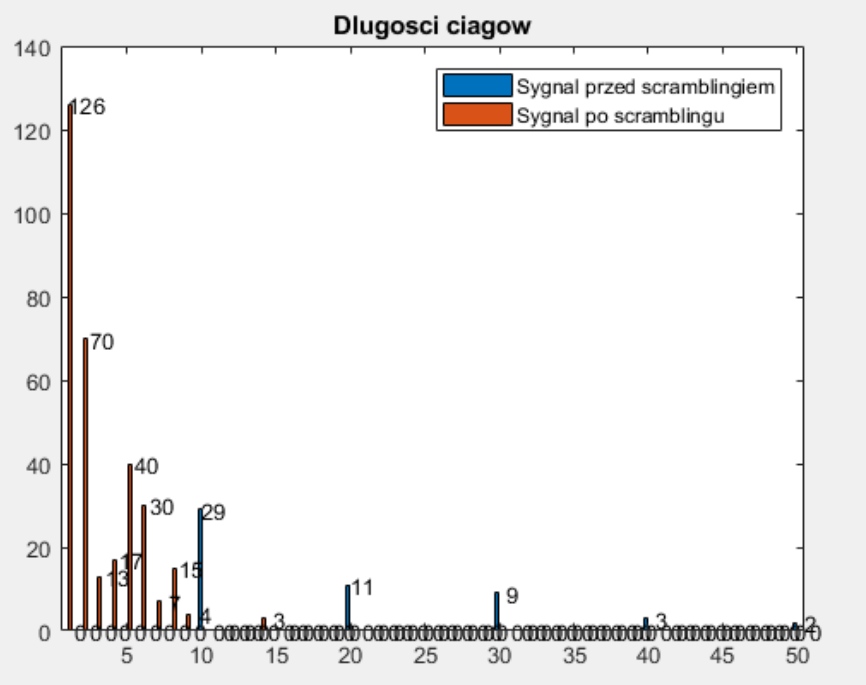
Symulator znajduje się w repozytorium https://github.com/kekusss/NiDUC

Obserwacje:

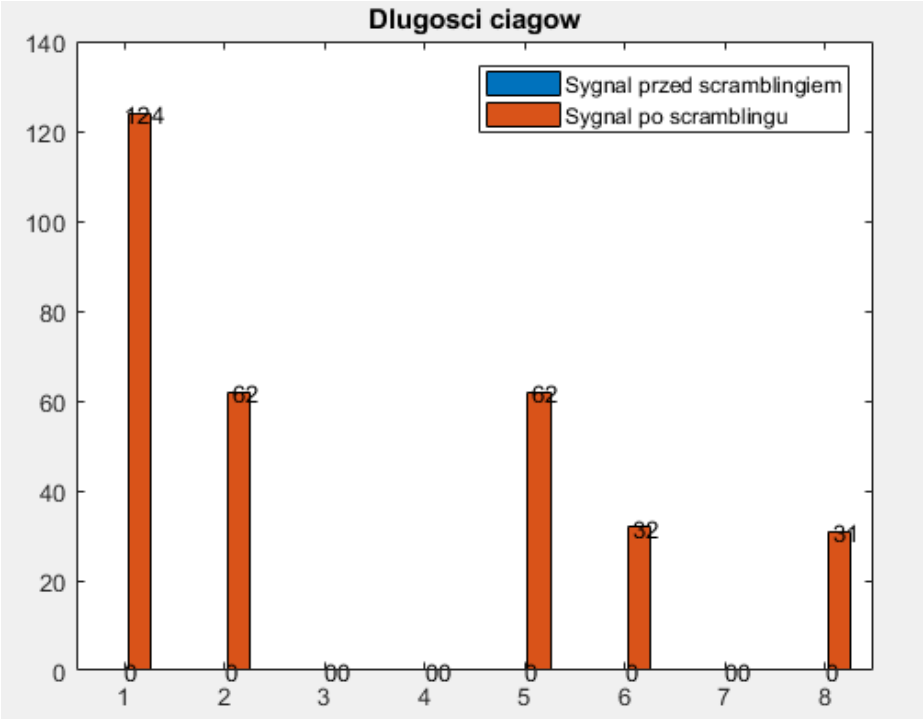
Długość słowa wejściowego: **1000 bitów**

Scrambler addytywny DVB:

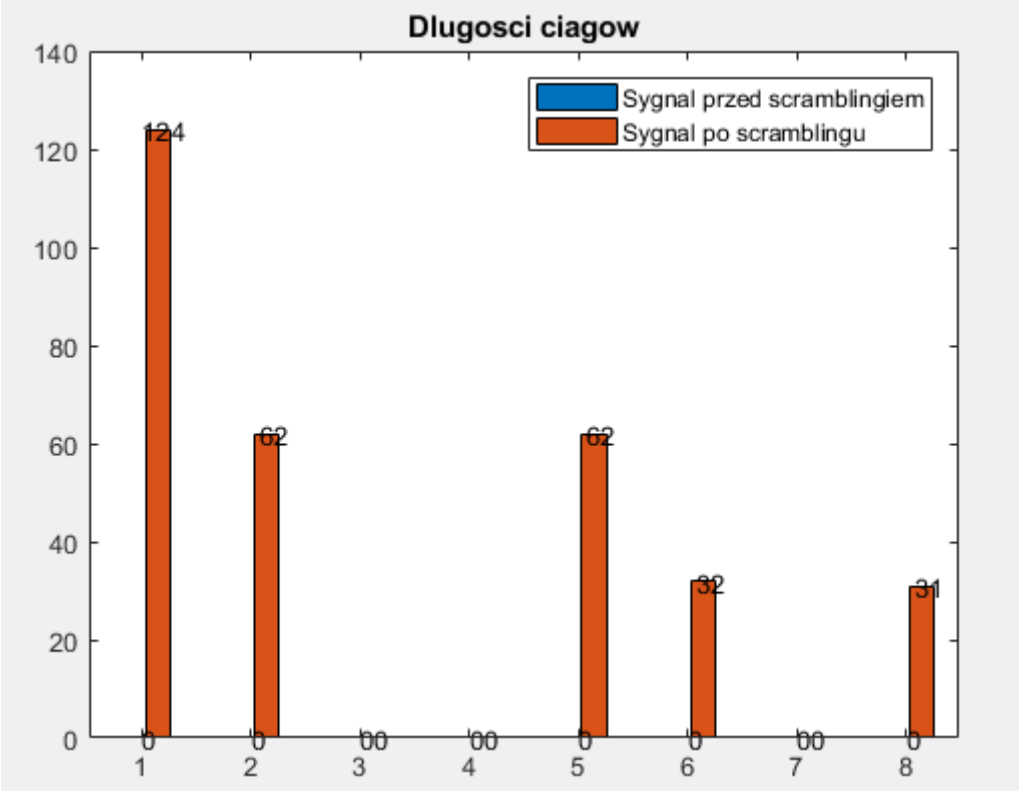
- dla sygnału pseudolosowego:



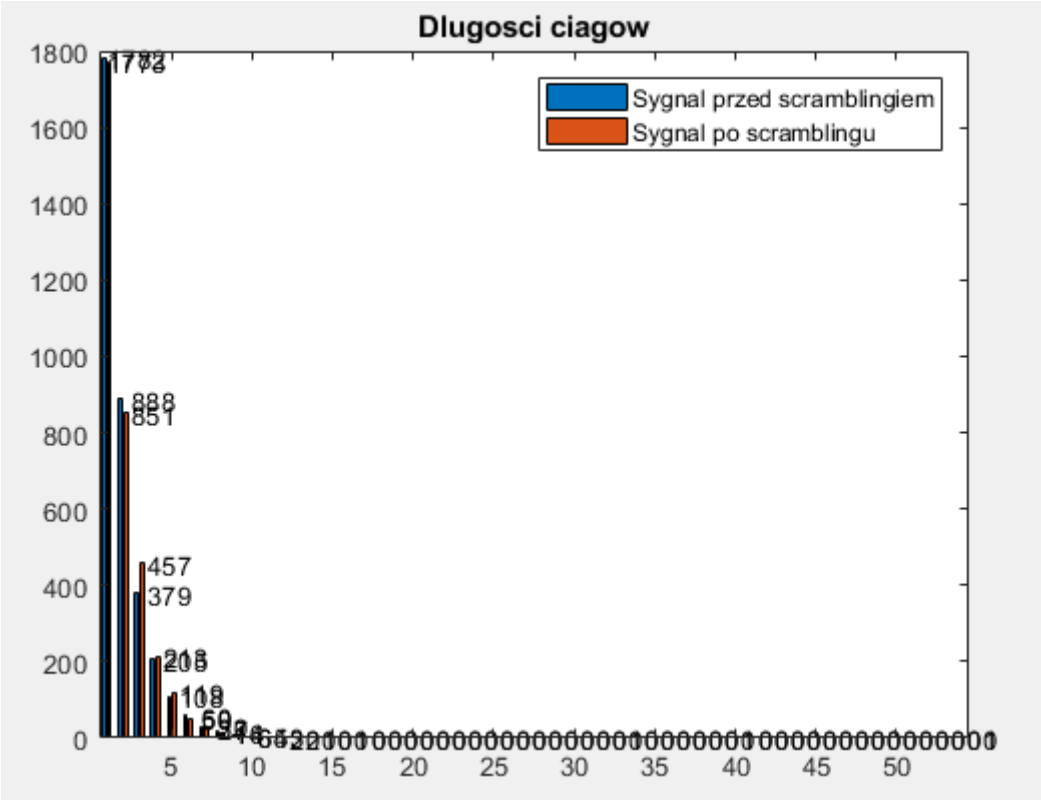
- dla sygnału jedynek:



- dla sygnału zer:

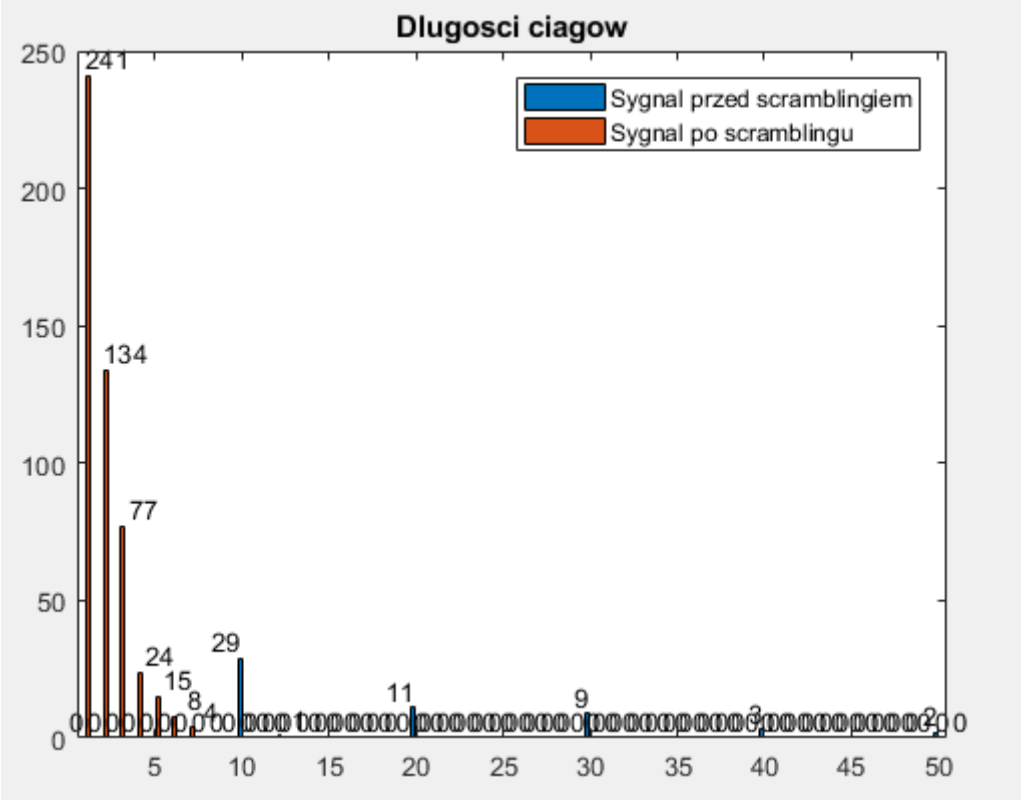


- Dla sygnału odczytanego z pliku

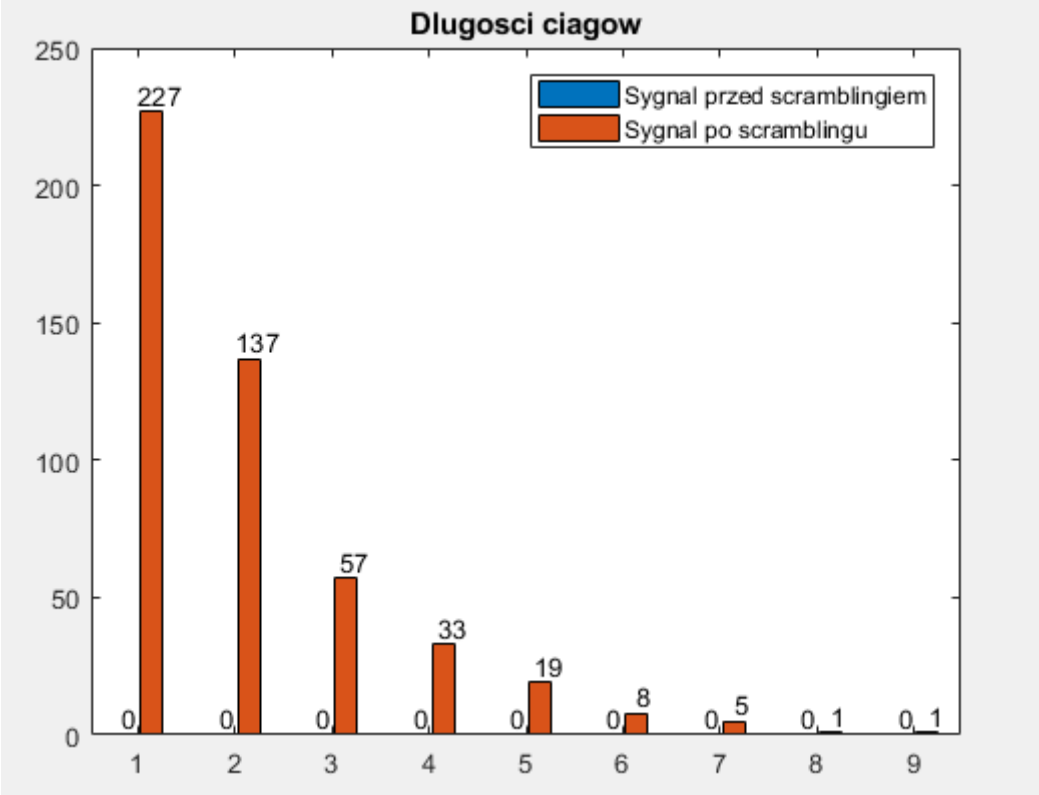


Scrambler multiplikatywny V.34

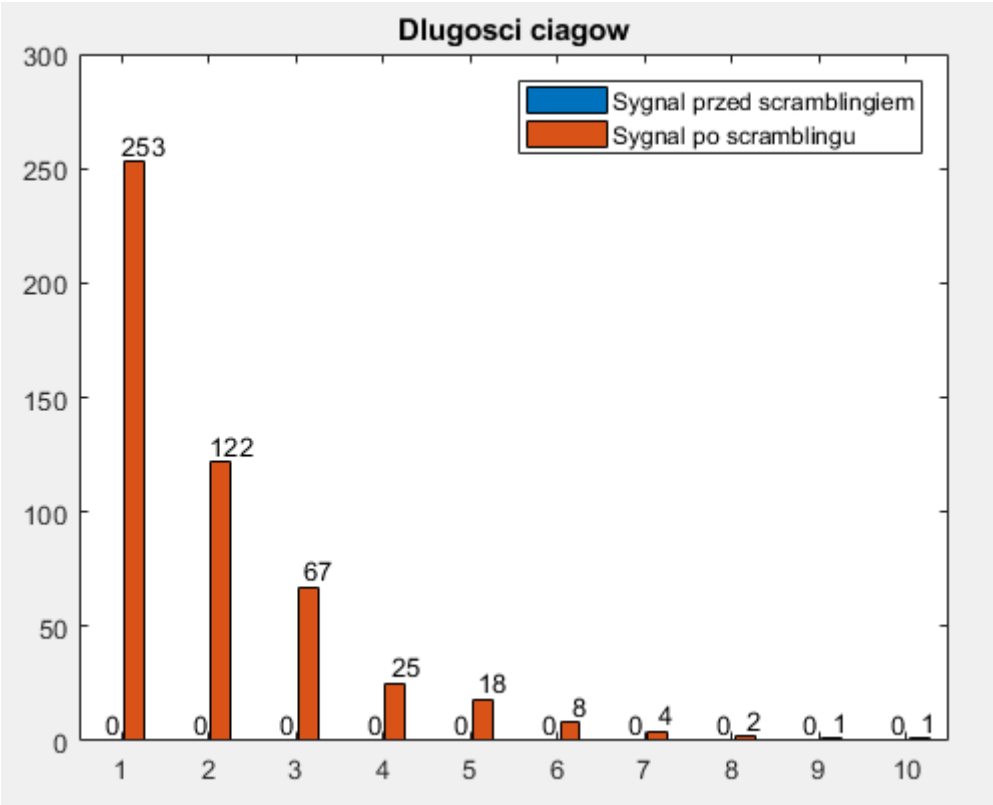
- dla sygnału losowego:



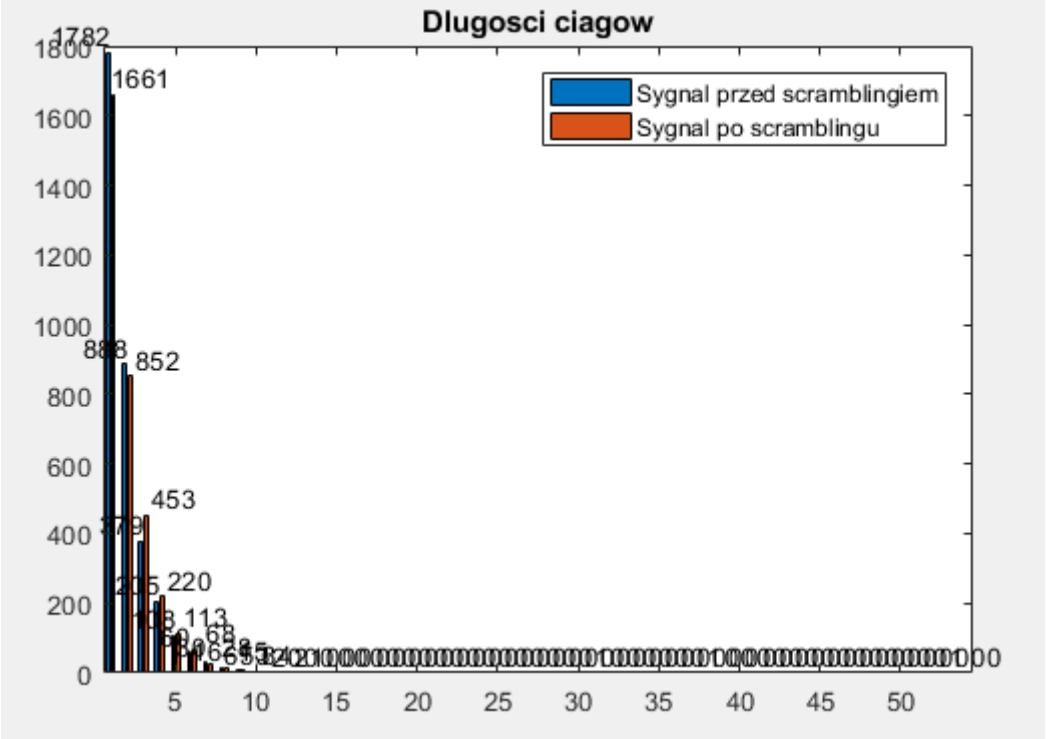
- dla sygnału jedynek:



- dla sygnału zer:

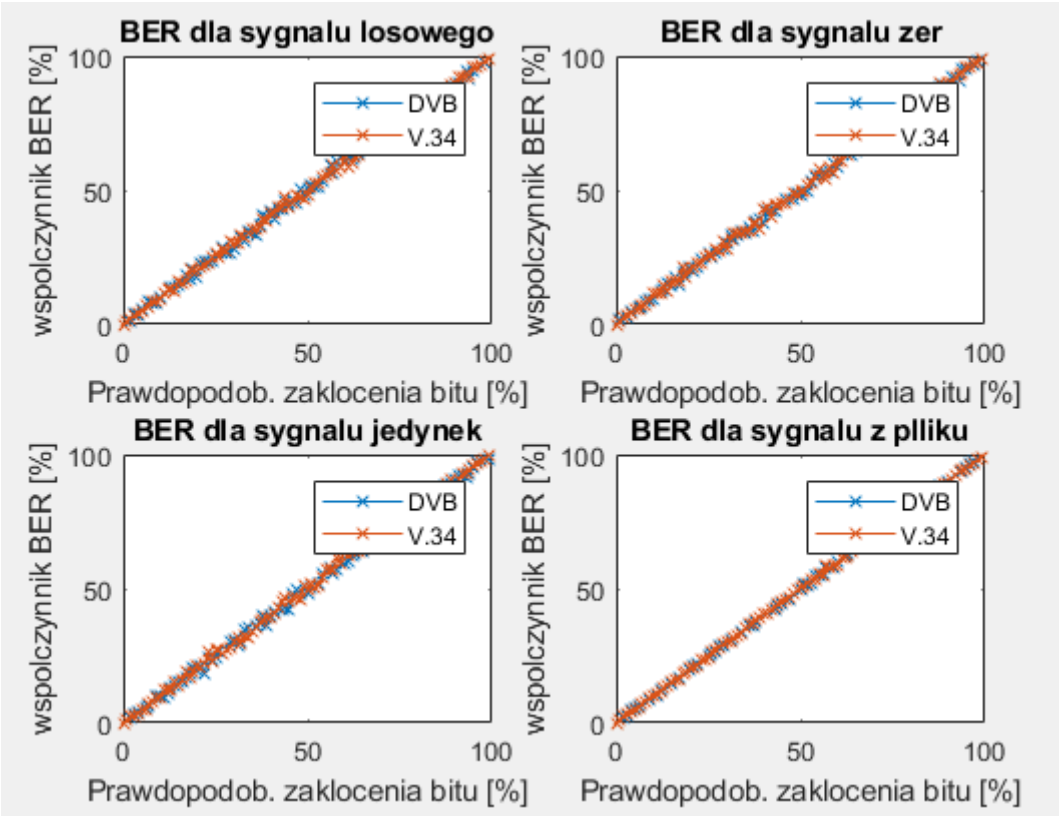


- Dla sygnału odczytanego z pliku

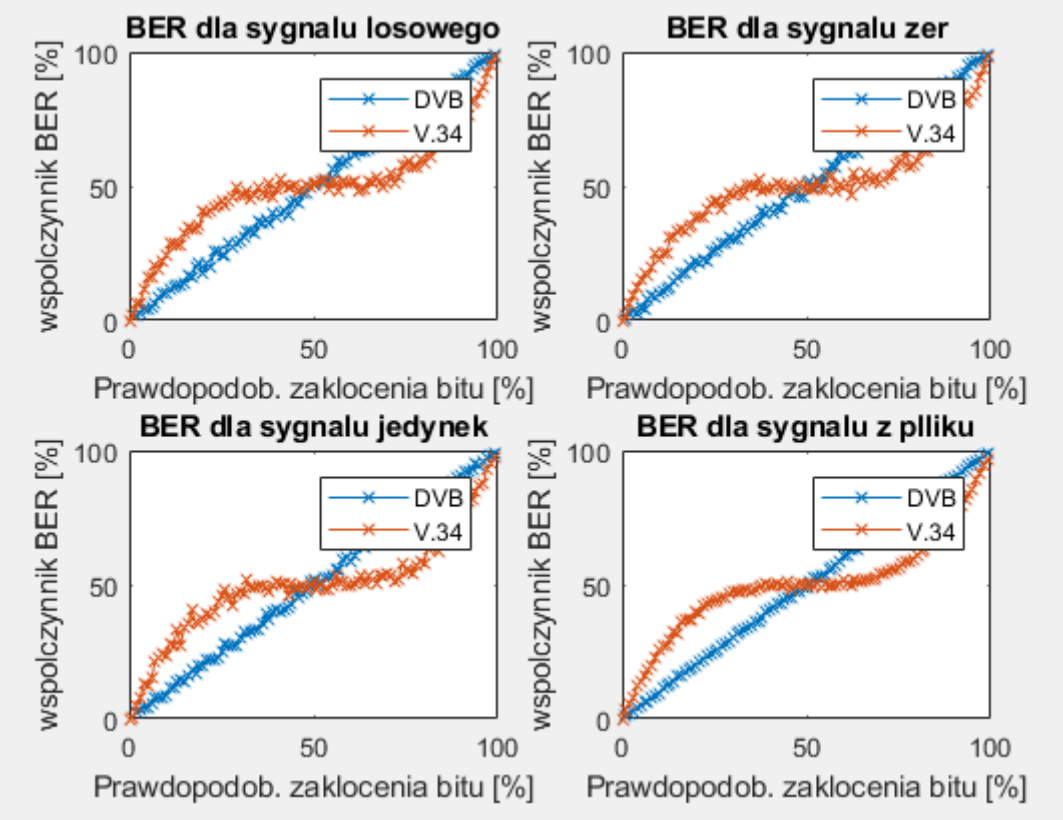


- BER:

- Bez scramblingu:



- Z scramblingiem:



Analiza wyników

W bezpośrednim porównaniu scrambler multiplikatywny (V.34) wypada lepiej od addytywnego (DVB). Dla danych pseudolosowych, ciągów samych zer oraz samych jedynek ilość tworzonych ciągów o mniejszej długości jest większa. Jeśli chodzi o dane z pliku jpg efektywność działania scramblerów jest podobna, wyniki były bardzo zbliżone. Analizując wykresy współczynnika przekłamania bitów (BER) można zauważyć, że bez zastosowania scramblingu współczynnik ten niemalże równa się prawdopodobieństwu przekłamania pojedynczego bitu dla obu systemów scramblingu niezależnie od sygnału. Zaobserwowano, że przy użyciu scramblera DVB, BER zmienia się podobnie jak bez stosowania scramblingu. Natomiast dla systemu V34, BER przy prawdopodobieństwie zakłócenia pojedynczego bitu mniejszym niż 50% jest większy niż bez scramblingu, gdzie przy większym prawdopodobieństwie współczynnik ten maleje.

Wnioski

Zrealizowanie projektu pomogło zrozumieć nam zasadę działania szyfratorów mieszających addytywnych jak i multiplikatywnych. Poznaliśmy ideę przesyłania danych kanałem komunikacyjnym oraz zapoznaliśmy się z pojęciem Bit Error Rate, które obrazuje przekłamania występujące podczas przesyłania strumienia danych. Ponadto, analiza wyników otrzymanych z eksperymentu uświadomiła nas jakie są różnice wynikające z korzystania z innych rodzajów scramblerów.