|  |  |
| --- | --- |
| **POLITECHNIKA WROCŁAWSKA**  **wydział Elektroniki**  **kierunek Automatyka i Robotyka**  **E - media** | |
| Data wykonania | Tytuł |
| 29.05.2018 r. | **Sprawozdanie z projektu** |
| Grupa |
| wtorek, godzina 9:15 |
| Autorzy | Aleksander Pawlak 226318, Mateusz Szydłowski 226408 |

1. **Cel Projektu**

Celem projektu było przygotowanie aplikacji dekodującej informacje zawarte w nagłówku pliku dźwiękowego, w formacie Wave.

Pokazać atrybuty pliku (rozmiar, głębię koloru, częstotliwość próbkowania, itp.). Następnie wyświetlić wykres widma fragmentu pliku za pomocą szybkiej transformaty fouriera (FFT).

Wykonać moduły (a) szyfrujące i (b) deszyfrujące plik Wave. Wykorzystać metodę szyfrowania asymetrycznego RSA w wersji 8 i 128 bitowej oraz szyfrowanie XOR. Zaszyfrować wyłącznie masę bitową pliku, pozostawiając nagłówek bez zmian - plik zaszyfrowany musi się dać otworzyć standardowymi aplikacjami, ale jego zawartość powinna być zakodowana (zaszyfrowana). Zaimplementować moduł deszyfrujący. Ocenić, czy informacje są możliwe od odczytania po zaszyfrowaniu (np. Odtworzenie pliku lub wyświetlenie wykresu wartości).

1. **Wykonanie**

Realizacją projektu jest program zaimplementowany w języku C++ z wykorzystaniem bibliotek SFML oraz boost. Pierwsza biblioteka wykorzystana została do rysowania wykresów FFT sygnału dźwiękowego przechowywanego przez plik oraz wykresów samego sygnału, biblioteka boot użyta została do operacji na liczbach powyżej 64 bit, które nie są oferowane przez bibliotekę standardową C++. Interfejs użytkownika stanowi proste w obsłudze menu w terminalu.

* **Wczytanie pliku:**

Program wczytuje w pierwszej kolejności nagłówek pliku wav do struktur zaimplementowanych na podstawie dokumentacji formatu wav. Część RIFF oraz fmt nagłówka wczytywane są do struktury Wav\_Header, natomiast początkowe dane z częśći data wczytywane są do struktury Wav\_Data.

Dane na ten temat zaczerpnięte zostały ze strony http://soundfile.sapp.org/doc/WaveFormat/.

Po wczytaniu nagłówka dane z próbkami wczytywane są do wektora zmiennych typu short int czyli 16 bitowych.

* **FFT próbek dźwięku:**

Program poza odczytaniem pliku wylicza dyskretną Transformatę Fouriera za pomocą algorytmu fft Cooleya-Tukeya.

Algorytm został sprowadzony do rekurencyjnej funkcji upraszczającej implementację algorytmu. Dane zwracane są w postaci wektora liczb zespolonych.

Po wyliczeniu transformaty Fouriera dla danych, wartości z wektora wynikowego wyświetlane są na ekranie w dodatkowym okienku.

Zrealizowany został również algorytm odwrotnej fft oparty na wcześniejszym algorytmie, w celu sprawdzenia poprawności wykonania samego algorytmu fft.

Dane po nałożeniu po sobie algorytmów zgadzały się w większym stopniu z danymi wejściowymi.

* **Szyfrowanie danych dźwiękowych:**

Wszystkie algorytmy szyfrujące wraz z funkcjami pomocniczymi zostały zaimplementowane w namespace EncryptionAlgorithms.

Pierwszym zrealizowanym algorytmem szyfrowania jest algorytm szyfrowania danych oparty na algorytmie XOR, który oprócz danych do szyfrowania wymaga wczytania klucza do szyfrowania. XOR zwraca na wyjściu 1, wtedy, gdy jeden i tylko jeden z bitów w danej zmiennej jest równy 1.

Algorytm wykorzystuje alternatywę binarną do której służy operator '^'.

Ten sam klucz potrzebny jest do szyfrowania i odszyfrowania danych. Postać klucza w programie jest zdefiniowana jako wektor o długości zadawanej przez użytkownika, może mieć postać jednej liczby, wektora długości wektora danych lub kliku liczb które będą wykorzystywane w zapętleniu.

W ramach algorytmu RSA zaimplementowane zostały następujące metody:

* int EncryptionAlgorithms::powMod(int value, int pow, int m) – metoda docelowo szyfrująca wartość value poprzez wyliczenie wartości według wzoru:

Gdzie c jest wynikiem, t jest wartością liczby value, natomiast e odpowiada parametrami pow, jako który podawane są odpowiednie klucze, zaś n jest parametrem m funkcji, jako który podawany będzie moduł.

* RsaKeys generateKeys(int, int) - do generowania kluczy w postaci struktury trzech zmiennych dla podanych dwóch liczb pierwszych.
* std::vector<short int> encryptRsa8 (const std::vector<short int>& inputData, const short int & e, const short int & n) -

funkcja kodująca dane z użyciem algorytmu RSA, przyjmująca wektor zmiennych typu short int, czyli 16 bitowych, odpowiadających rozmiarowi standardowych sampli w pliku wav.

Zaimplementowana funkcja dzieli dane w wektorze na zmienne typu uint8\_t, czyli zmienne 8 bitowe, które są następnie szyfrowane do

zmiennych typu short int. Jest to związane z tym, że zakodowane wartości liczbowe będą większe od oryginalnych i mogą wychodzić poza zakres

typu uint8\_t. W związku z takim działaniem zwracany wektor jest dwa razy większy.

* std::vector<short int> decryptRsa8(const std::vector<short int>& inputData, const short int & e, const short int & n) -

funkcja dekodująca dane przy użyciu algorytmu RSA. Funkcja zakłada, że dane zakodowane są algorytmem encryptRsa8, wobec czego z wektora

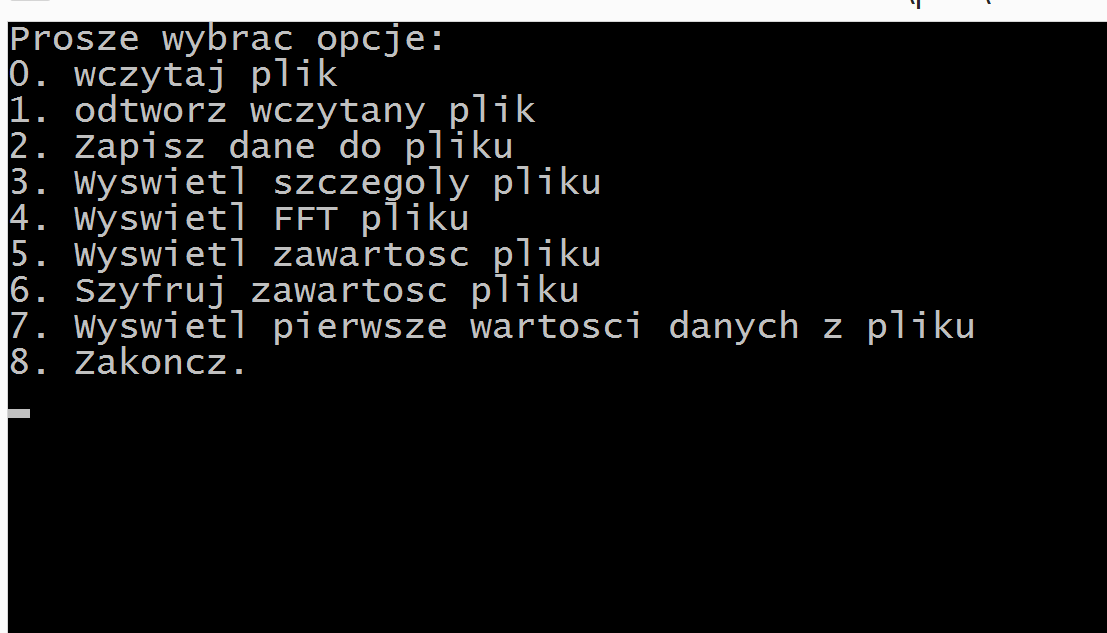
wejściowego pobiera po dwie zmienne, będące blokami po 16 bitow i koduje je do jednego bloku 16 bitowego. Wyjściowy wekto jest dwa razy mniejszy od wejściowego.

Do realizacji szyfrowania RSA w wersji 128 bitowej zaimplementowane zostały takie same metody operujące na większych liczbach. W przypadku metody szyfrującej różnica w stosunku do szyfrowania 8 bitowego polega na braniu bloków 128 bitowych, czyli 8 liczb z wektora danych i szyfrowaniu ich do bloku 256 bitowego.

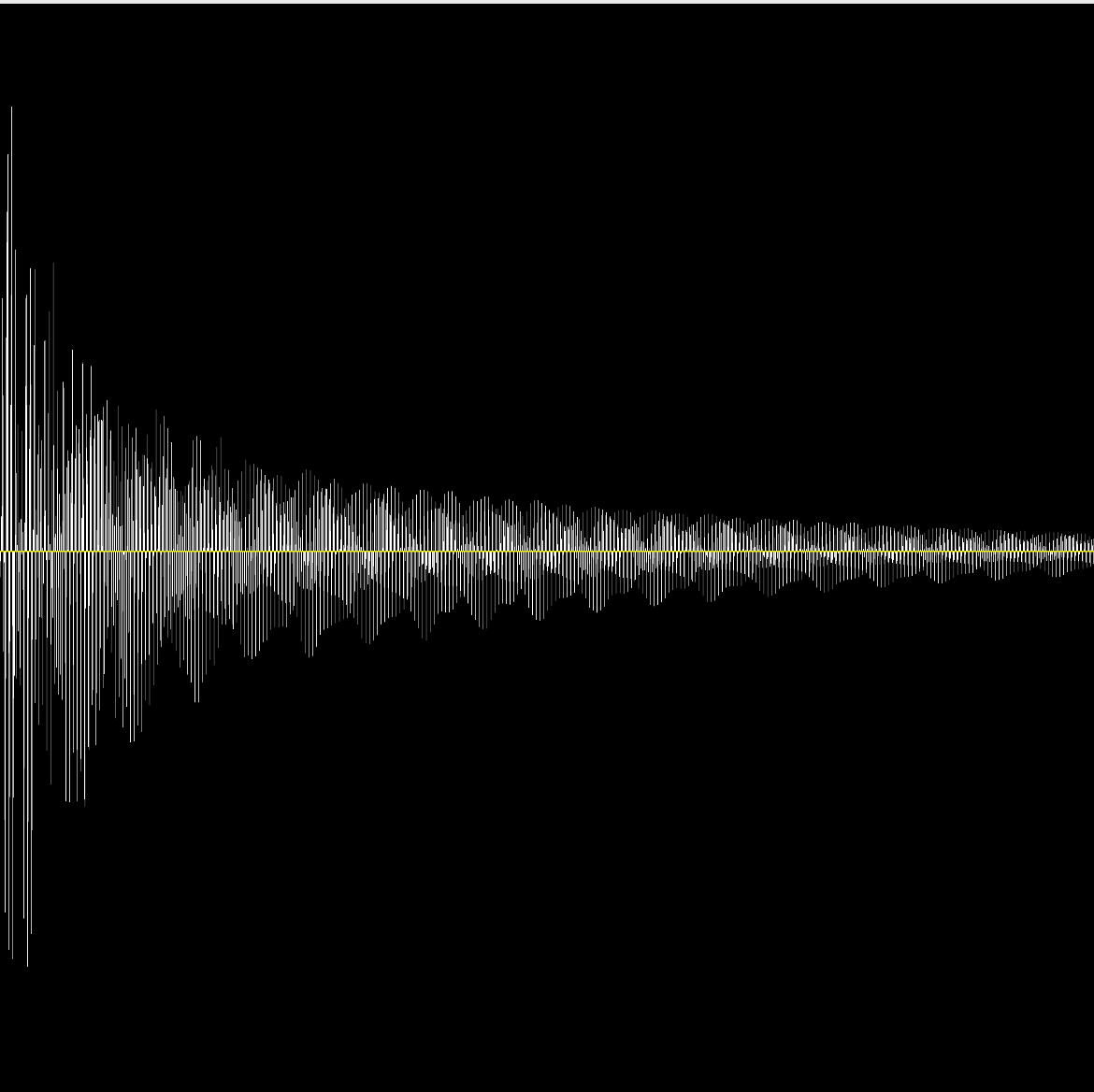
Metoda dekodująca deszyfruje bloki 256 bitowe do postaci 128 bitowych bloków pamięci.

Algorytm RSA wykorzystuje klucze wygenerowane na podstawie odpowiednio dużych liczb pierwszych, które mogą zostać podane przez użytkownika w terminalu lub wczytane z odpowiedniego pliku. Zestaw potrzebnych wartości ma postać klucza publicznego – wykorzystywanego do szyfrowania, prywatnego – używanego do odszyfrowania oraz wartości modulo używanej podczas obydwóch operacji.

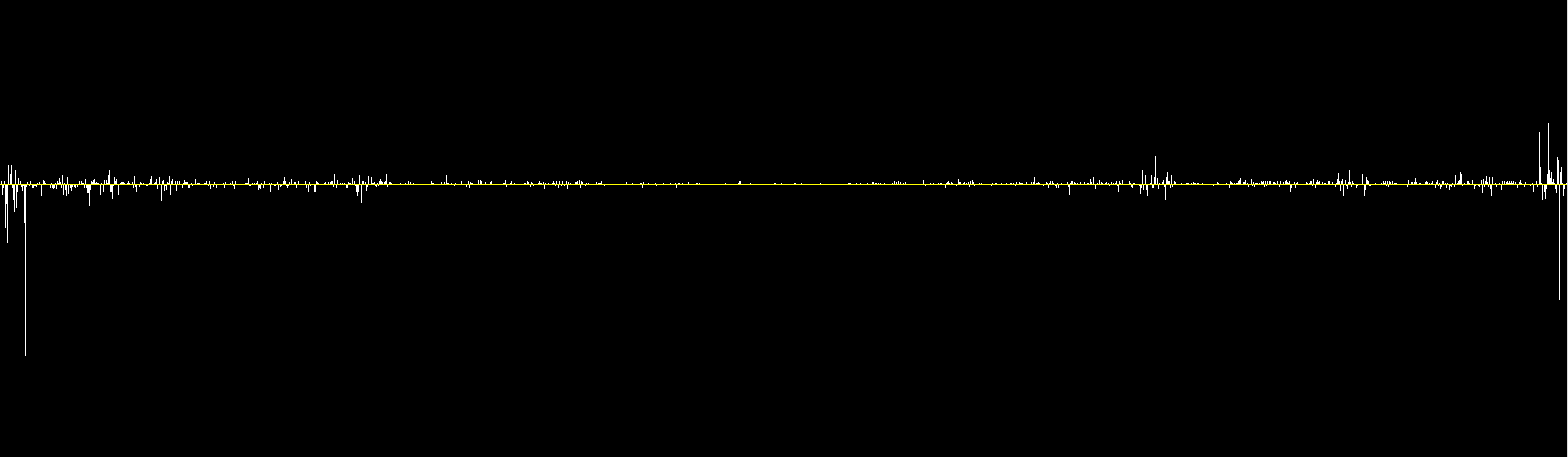
* **Wygląd menu aplikacji oraz testy zaimplementowanych metod:**
* Po uruchomieniu aplikacji ukazuje nam się proste menu w terminalu, którego widok znajduje się poniżej.



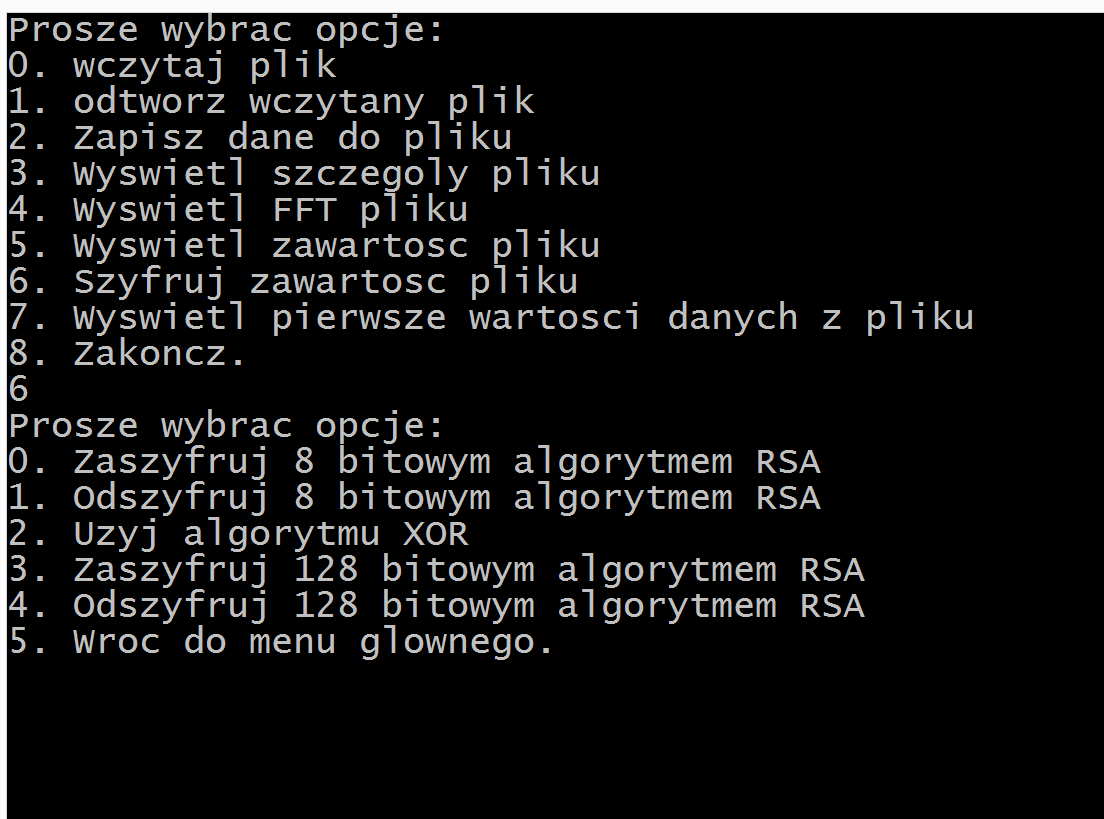
* Wyświetlany wykres sygnału wygląda następująco.



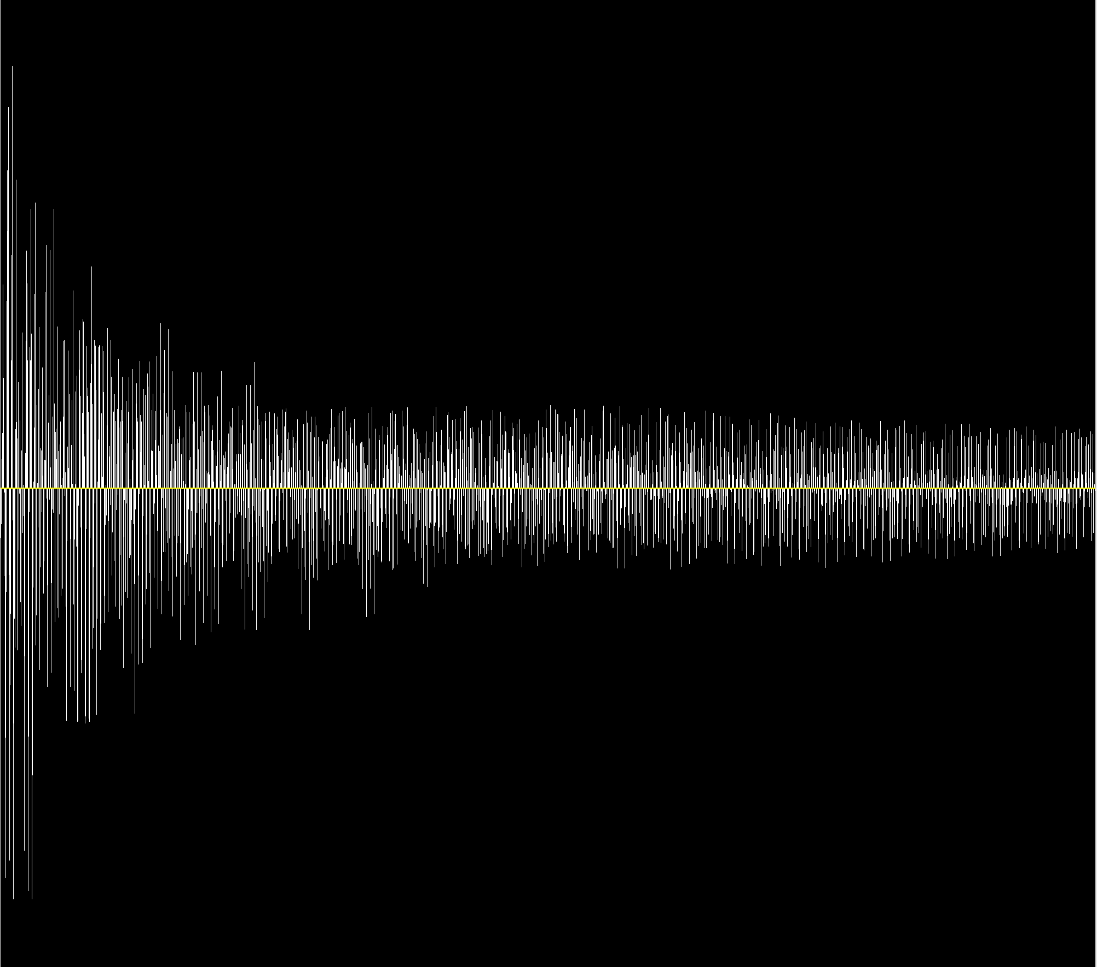
* Wykres FFT sygnalu.

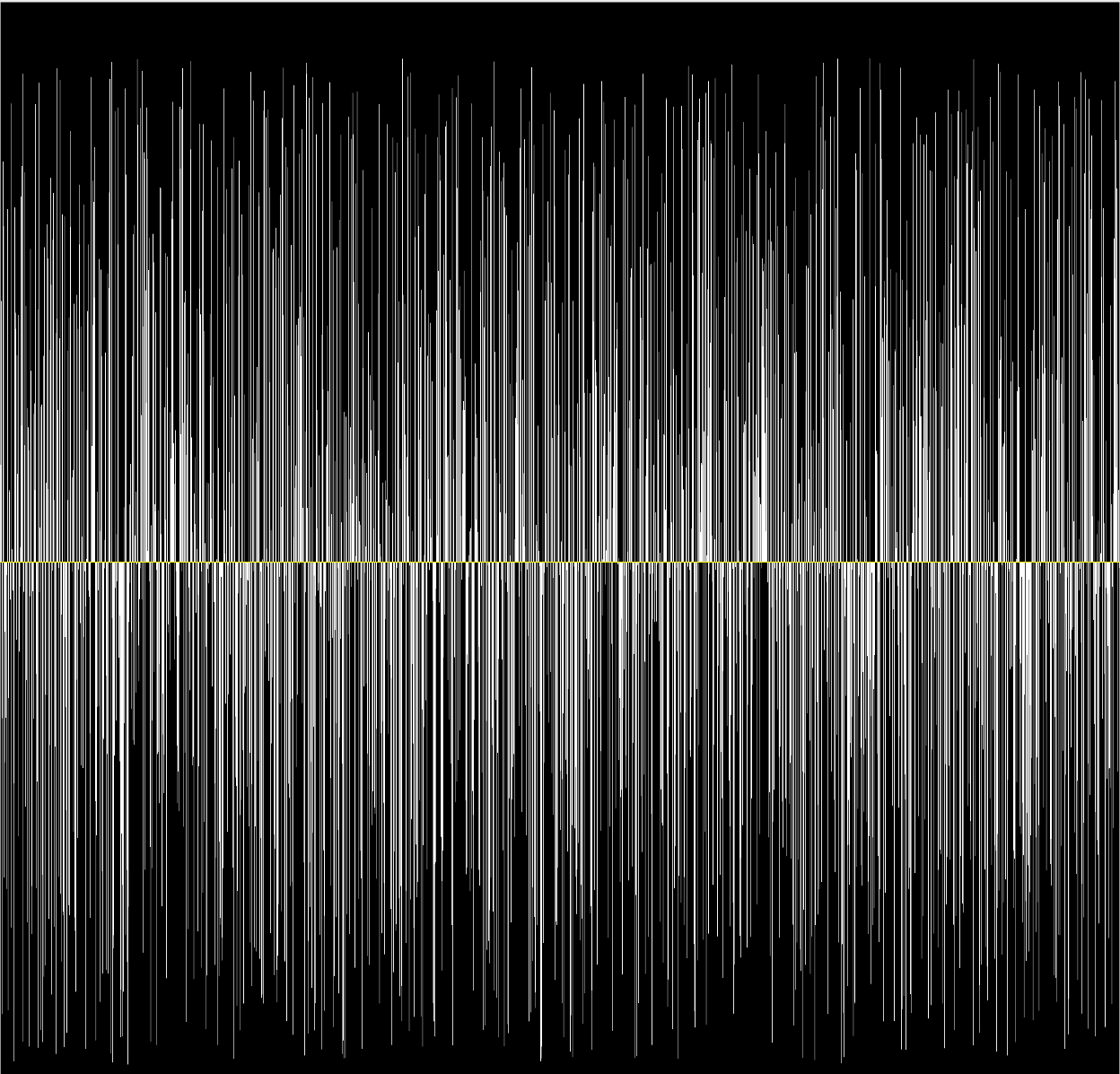


* Podmenu szyfrowania.



* Dane zaszyfrowane algorytmem XOR.



* Dane zaszyfrowane algorytmem RSA.

1. **Wnioski.**

Testy zaimplementowanych algorytmów wykazały poprawność ich działania.

Z trzech wykorzystanych metod szyfrowania danych prezentują się one następująco, w kolejności od najprostszego i zarazem najmniej bezpiecznego algorytmu do najbardziej złożonego obliczeniowo i trwającego więcej, a także dającego największe bezpieczeństwo:

Algorytm XOR, RSA – 8 bit, RSA – 128 bit.

Najlepszą metodę stanowi szyfrowanie algorytmem RSA, z czego 128 bitowe daje największe bezpieczeństwo ze względu na klucze o bardzo dużych wartościach, przez co możliwych liczb pierwszych służących do wygenerowania ich jest znacznie więcej, a co za tym idzie znalezienie ich zajmie o wiele więcej czasu niż znalezienie wartości 8 bitowych. Wadą takiego rozwiązania jest złożoność obliczeniowa, przez co szyfrowanie danych na przeciętym komputerze domowym zajmuje zauważalnie więcej czasu.

Warto również zaznaczyć, iż wybrany język programowania C++ jest jednym z lepszych do implementacji wymienionych algorytmów i samego wczytywania plików bitowych wav ze względu na jego tzw. niskopoziomowe metody z biblioteki standardowej, oraz lepszą wydajność przy złożonych obliczeniach niż jęzki wysokopoziomowe jak np. Python.