Dokumentacja projektu z przedmiotu Kryptografia i teoria kodów

Autor: Jakub Gwiżdż

19 grudnia 2024

Streszczenie

Opis ogólny celu aplikacji, który może obejmować zapewnienie bezpieczeństwa danych za pomocą szyfrowania, kompresji danych oraz podpisu cyfrowego. (Z wykorzystaniem schematyzmu struktury znaczników LaTeXstworzonego przez R. J. Wysockiego.)

Spis treści

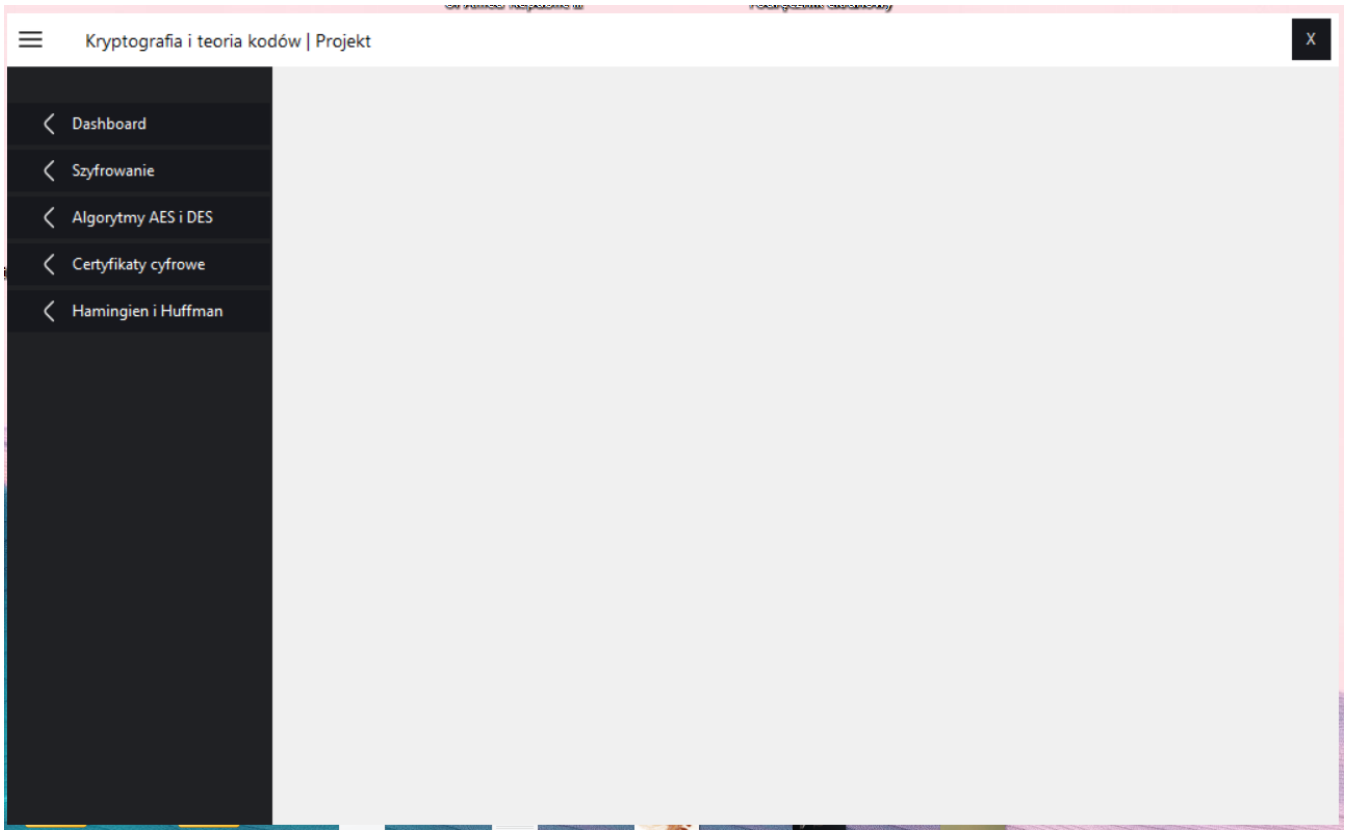
Spis tabel

Spis rysunków

1 Wstęp

Aplikacja zapewnia różne techniki szyfrowania i kodowania, które zapewniają poufność danych, integralność, oraz uwierzytelnienie. W aplikacji wykorzystano szyfry klasyczne, takie jak szyfr podstawieniowy, szyfr transpozycyjny, oraz nowoczesne algorytmy, takie jak AES i DES. Ponadto, obsługiwane są podpisy cyfrowe oraz techniki kodowania, jak Hamming i Huffman.

2 Architektura aplikacji



Rysunek 1: Schemat szynoprzewodów

Diagram architektury systemu: Wskazanie, jak różne moduły aplikacji (szyfrowanie, podpis cyfrowy, kompresja) współdziałają ze sobą. Opis struktury kodu: Krótkie przedstawienie struktury projektu (np. foldery, pliki odpowiedzialne za konkretne algorytmy).

3 Algorytmy szyfrowania

3.1 Prosty szyfr podstawieniowy

Prosty szyfr podstawieniowy to metoda szyfrowania, w której każda litera tekstu jawnego jest zastępowana inną literą zgodnie z ustalonym kluczem. Jest to jedna z najprostszych technik kryp-

tograficznych, charakteryzująca się niskim poziomem bezpieczeństwa.

3.1.1 Opis działania algorytmu

- **Definicja klucza szyfrującego:** Klucz to ustalone przypisanie liter alfabetu. Na przykład:
  - Alfabet jawny: `A B C D E F G H I J K L M N O P Q R S T U V W X Y Z`
  - Klucz szyfrujący: `Q W E R T Y U I O P A S D F G H J K L Z X C V B N M`
- **Szyfrowanie:** W procesie szyfrowania każda litera tekstu jawnego jest zastępowana literą odpowiadającą jej w kluczu szyfrującym. Na przykład: `HELLO → ITSSG`.
- **Deszyfrowanie:** Proces deszyfrowania jest odwrotnością szyfrowania i wykorzystuje klucz odwrotny.

3.1.2 Funkcjonalność w aplikacji

Aplikacja oferuje następujące funkcje:

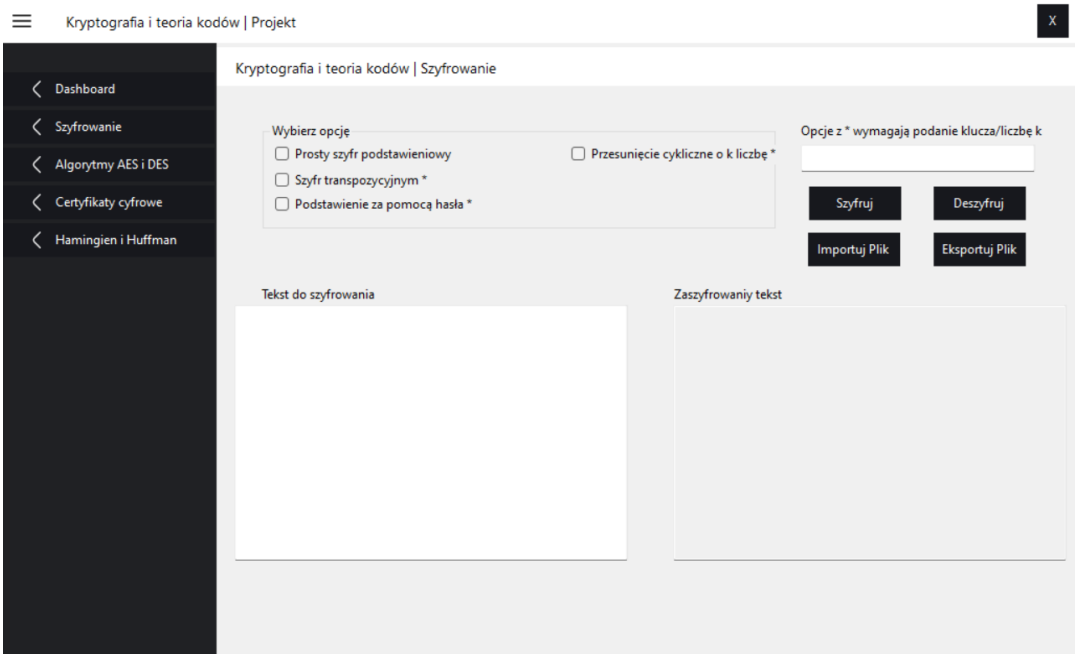
- Pole do wprowadzania tekstu jawnego.
- Pole do wyświetlania szyfrogramu po zaszyfrowaniu tekstu.
- Możliwość deszyfrowania tekstu.

3.1.3 Przykłady działania

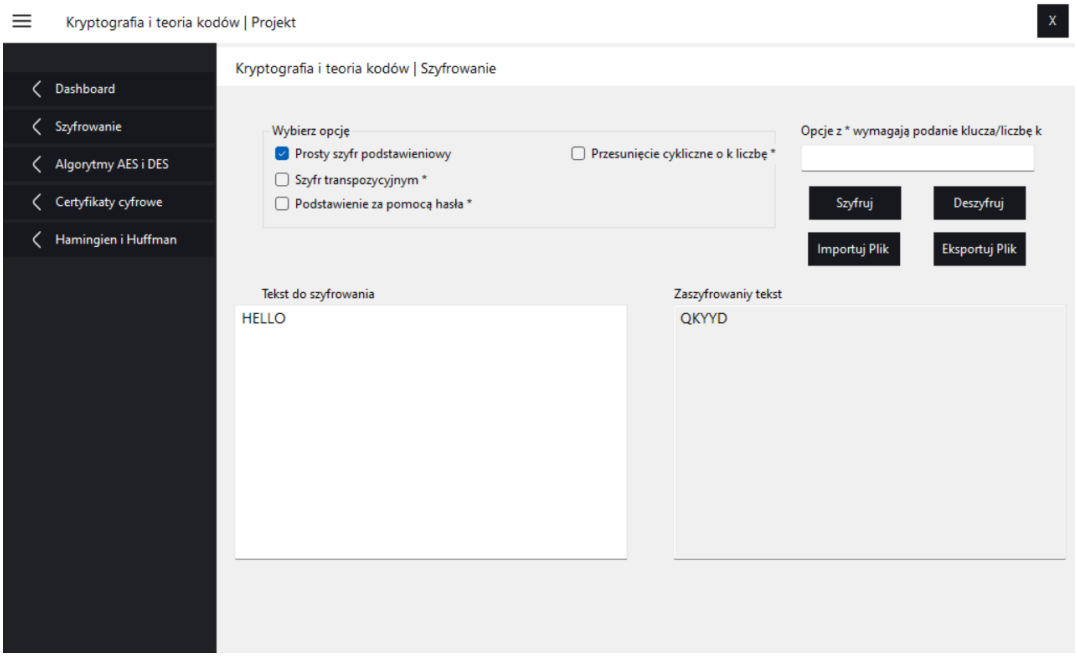
Tekst jawny	**Klucz szyfrujący**	**Szyfrogram**
HELLO	QWERTYUIOPASDFGHJKLZXCVBNM	ITSSG
WORLD	MNBVCXZLKJHGFDSAPOIUYTREWQ	NDKAW
CRYPTO	ZYXWVUTSRQPONMLKJIHGFEDCBA	XILKGJ

Tabela 1: Przykłady szyfrowania prostym szyfrem podstawieniowym

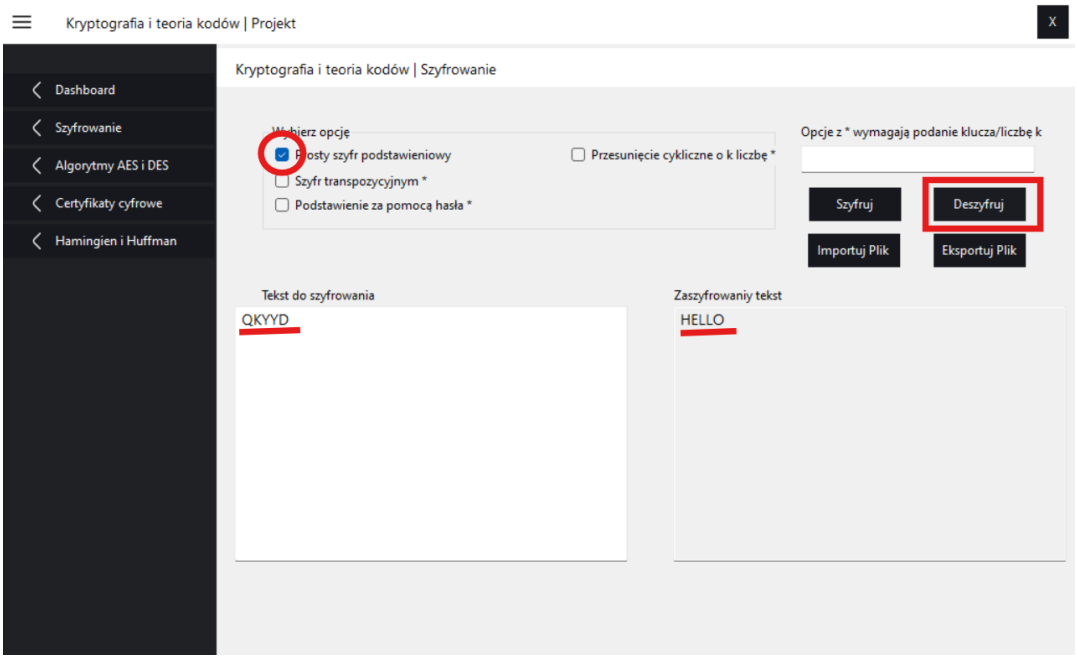
3.1.4 Przykładowe ekrany aplikacji



Rysunek 2: Pole wprowadzania tekstu jawnego i klucza szyfrującego.



Rysunek 3: Wynik szyfrowania – wyświetlenie szyfrogramu.



Rysunek 4: Proces deszyfrowania – wprowadzenie szyfrogramu i wyświetlenie tekstu jawnego.

3.1.5 Wady i zalety

- **Zalety:**
  - Prosta implementacja.
  - Intuicyjność działania.
- **Wady:**
  - Łatwość złamania przy użyciu analizy częstotliwości.
  - Niski poziom bezpieczeństwa.

3.2 Szyfr transpozycyjny

Szyfr transpozycyjny to metoda szyfrowania, która polega na przestawianiu znaków tekstu jawnego według określonego schematu, zamiast ich zamiany na inne znaki. Tego typu algorytmy wykorzystują zasadę zmiany kolejności znaków, zachowując oryginalne litery.

3.2.1 Opis działania algorytmu

- **Definicja klucza szyfrującego:** Klucz może być liczbą całkowitą, reprezentującą liczbę kolumn, lub bardziej złożoną strukturą opisującą sposób transpozycji.
- **Szyfrowanie:**
  1. Tekst jawny jest umieszczany w tabeli o określonej liczbie kolumn.
  2. Znaki są odczytywane kolumnami w określonej kolejności, zgodnie z kluczem.
- **Deszyfrowanie:** Proces odwrotny, w którym szyfrogram jest układany w tabeli i odczytywany wierszami.

3.2.2 Funkcjonalność w aplikacji

W aplikacji dostępne są następujące opcje:

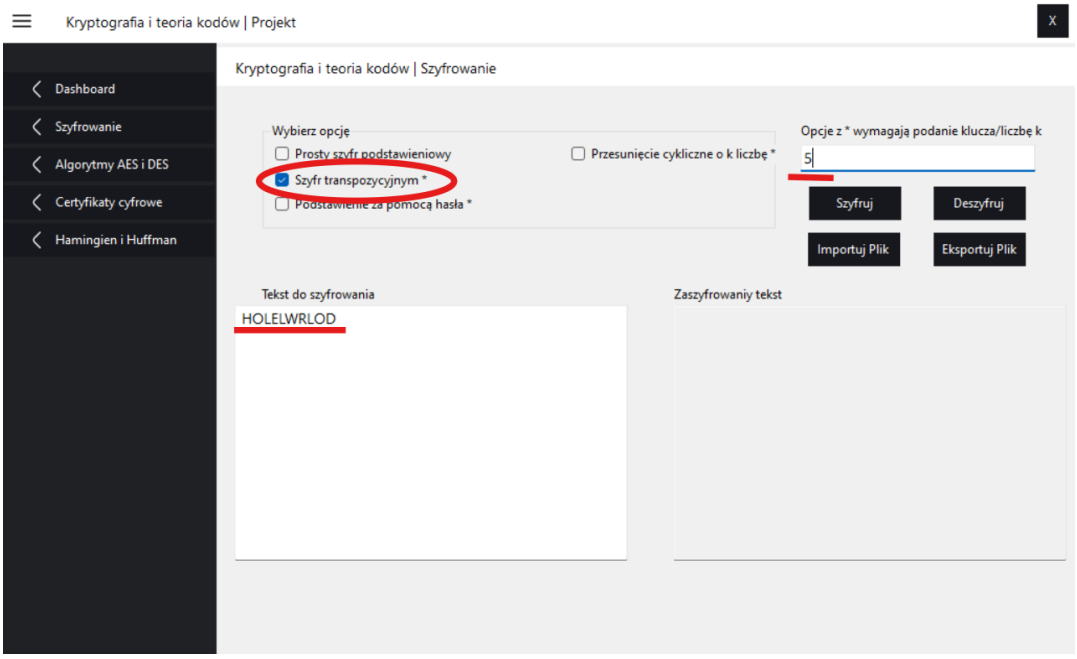
- Wprowadzenie tekstu jawnego do zaszyfrowania.
- Definicja klucza szyfrującego (np. liczba kolumn w tabeli).
- Generowanie tabeli z podziałem na kolumny dla wizualizacji procesu szyfrowania.
- Wyświetlenie wynikowego szyfrogramu.
- Deszyfrowanie szyfrogramu z użyciem klucza.

3.2.3 Przykłady działania

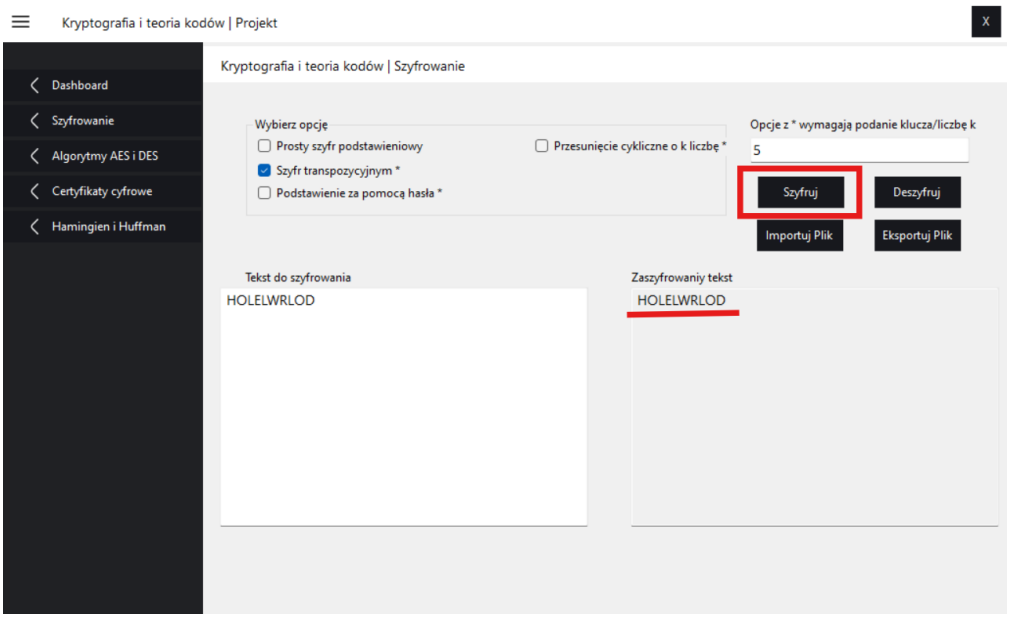
Tekst jawny	**Liczba kolumn**	**Tabela transpozycji**	**Szyfrogram**
HELLO WORLD	5	H E L L O W O R L D	HOLELWRLOD
CRYPTOGRAPHY	4	C R Y P T O G R A P H Y	CTARYROPYGP

Tabela 2: Przykłady szyfrowania szyfrem transpozycyjnym

3.2.4 Przykładowe ekrany aplikacji



Rysunek 5: Pole wprowadzania tekstu jawnego i klucza szyfrującego.



Rysunek 6: Wyświetlenie szyfrogramu po zaszyfrowaniu.

3.2.5 Wady i zalety

- **Zalety:**

  - Trudniejsze do złamania niż szyfry podstawieniowe.
  - Możliwość łączenia z innymi metodami szyfrowania dla zwiększenia bezpieczeństwa.
- **Wady:**
  - Wymaga bardziej złożonej implementacji niż prosty szyfr podstawieniowy.
  - Klucz musi być precyzyjnie znany, aby odczytać tekst zaszyfrowany.

3.3 Podstawienie za pomocą hasła

Szyfr podstawieniowy za pomocą hasła to metoda szyfrowania, w której klucz szyfrujący jest generowany na podstawie dostarczonego hasła. W tej technice litery tekstu jawnego są zastępowane innymi znakami według zdefiniowanej permutacji alfabetu, która wynika z hasła.

3.3.1 Opis działania algorytmu

Algorytm działania szyfru opiera się na kilku krokach. Po pierwsze, hasło jest wykorzystywane do permutacji alfabetu, przy czym usuwane są powtarzające się znaki. Następnie pozostałe litery alfabetu są dodawane w kolejności alfabetycznej, co prowadzi do utworzenia klucza szyfrującego. W procesie szyfrowania każda litera tekstu jawnego jest zastępowana literą z klucza odpowiadającą jej pozycji w alfabecie, a znaki niealfabetyczne pozostają niezmienione. Deszyfrowanie to odwrotna operacja, w której szyfrogram jest przekształcany z powrotem do tekstu jawnego na podstawie klucza.

3.3.2 Funkcjonalność w aplikacji

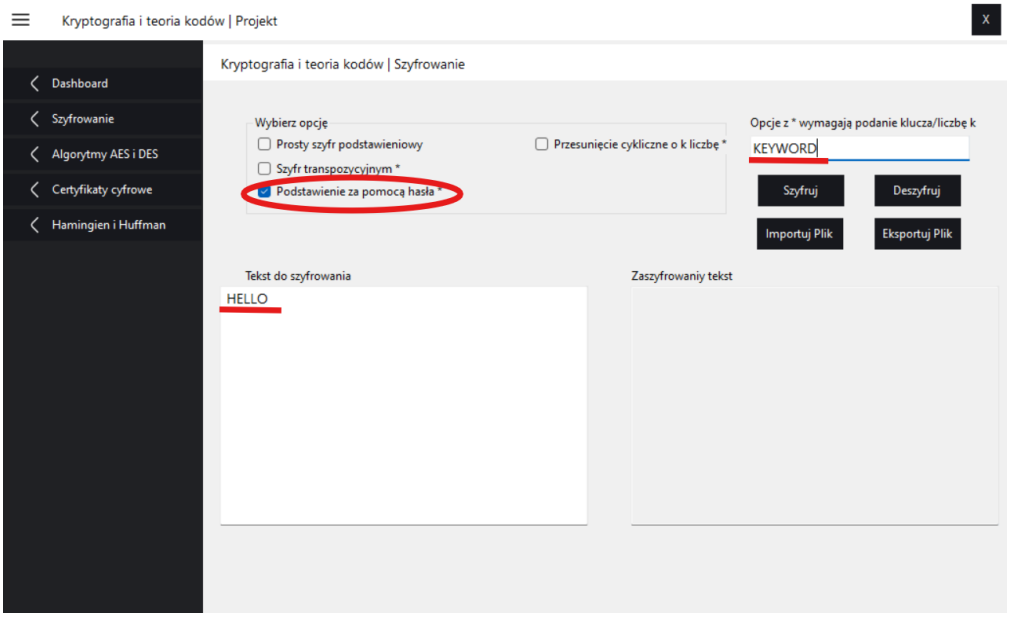
W aplikacji użytkownik ma możliwość wprowadzenia tekstu jawnego do zaszyfrowania oraz zdefiniowania hasła, które będzie wykorzystane do generowania klucza szyfrującego. Po wygenerowaniu klucza użytkownik może go zobaczyć oraz uzyskać szyfrogram po zaszyfrowaniu tekstu jawnego. Aplikacja umożliwia również odszyfrowanie szyfrogramu przy użyciu tego samego hasła.

3.3.3 Przykłady działania

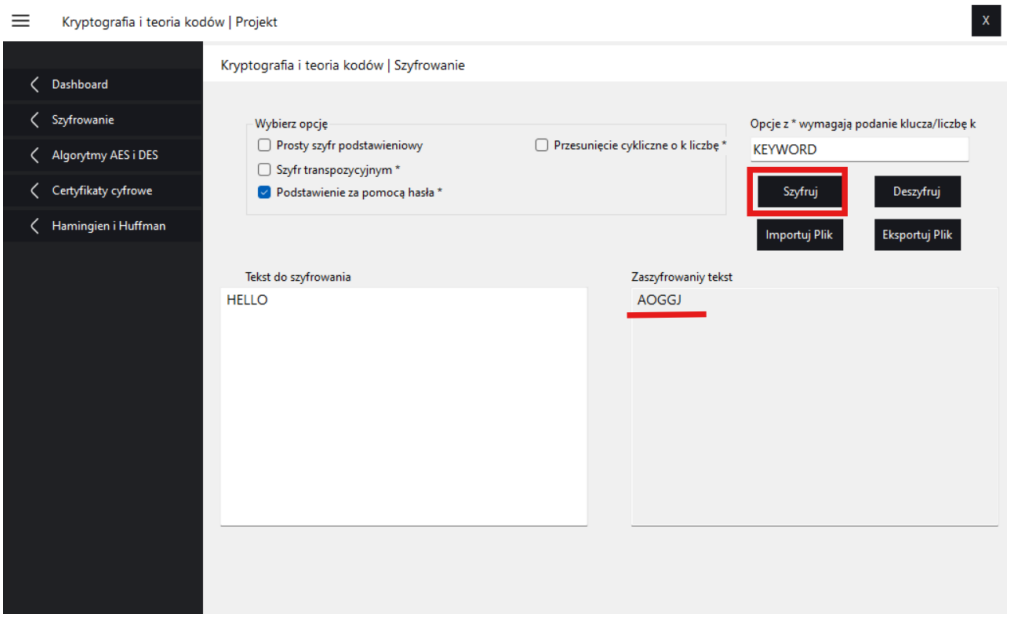
Tekst jawny	**Hasło**	**Klucz szyfrujący**	**Szyfrogram**
HELLO	KEYWORD	K, E, Y, W, O, R, D, A, B, C, F, G, H, I, J, L, M, N, P, Q, S, T, U, V, X, Z	AOGGJ
CRYPTOGRAPHY	PASSWORD	P, A, S, W, O, R, D, B, C, E, F, G, H, I, J, K, L, M, N, Q, T, U, V, X, Y, Z	AQSTWKLPQRO

Tabela 3: Przykłady szyfrowania za pomocą hasła

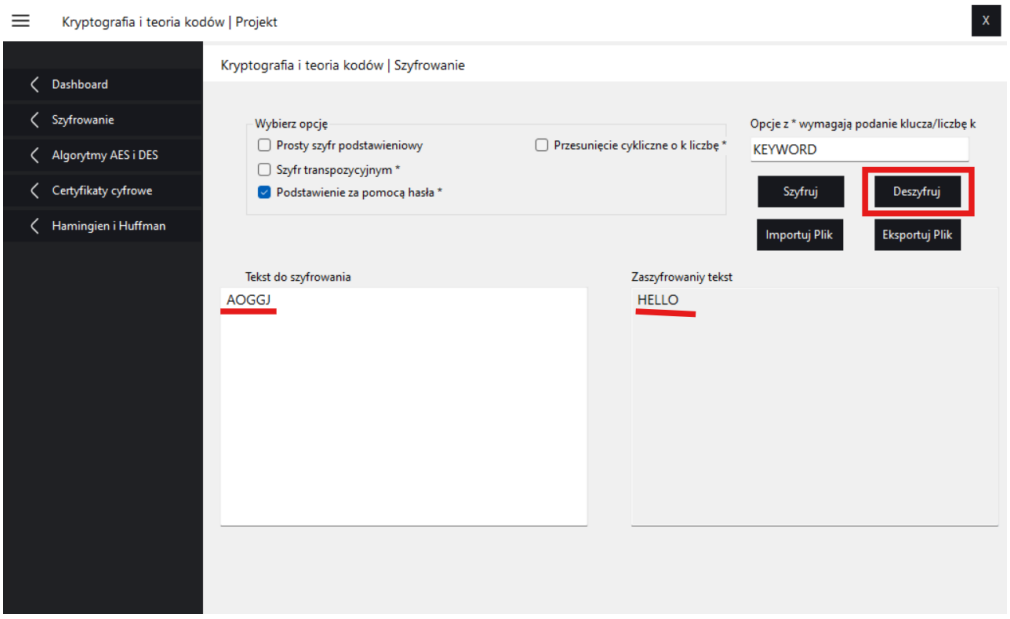
3.3.4 Przykładowe ekrany aplikacji



Rysunek 7: Pole wprowadzania tekstu jawnego i hasła szyfrującego.



Rysunek 8: Wyświetlenie szyfrogramu po zaszyfrowaniu.



Rysunek 9: Wyświetlenie szyfrogramu po deszyfrowaniu.

3.3.5 Wady i zalety

Szyfr podstawieniowy za pomocą hasła ma swoje zalety i wady. Do jego zalet należy prosta implementacja oraz szybkie działanie. Ponadto użycie hasła zwiększa bezpieczeństwo w porównaniu do klasycznego szyfru podstawieniowego. Z drugiej strony klucz jest zależny od hasła; jego słabość (na przykład łatwość odgadnięcia) może zmniejszać bezpieczeństwo całego systemu. Dodatkowo wymaga ono dokładnego przechowywania hasła dla celów deszyfrowania.

3.4 Przesunięcie cykliczne o k

Szyfrowanie z przesunięciem cyklicznym o k to jedna z prostych metod szyfrowania, w której każda litera tekstu jawnego jest przesuwana w alfabecie o stałą liczbę pozycji k. Po osiągnięciu końca alfabetu przesunięcie kontynuuje się od początku alfabetu, co można opisać jako operację modulo.

3.4.1 Opis działania algorytmu

Algorytm szyfrowania polega na kilku krokach. W pierwszej kolejności każda litera tekstu jawnego jest przesuwana w prawo o k pozycji w alfabecie. Jeśli przesunięcie przekroczy ostatnią literę alfabetu, wraca na początek (operacja modulo), a znaki niealfabetyczne pozostają niezmienione. Proces deszyfrowania jest odwrotny: każda litera szyfrogramu jest przesuwana w lewo o k pozycji. W przypadku, gdy przesunięcie wykracza poza początek alfabetu, wraca na jego koniec (operacja modulo).

3.4.2 Funkcjonalność w aplikacji

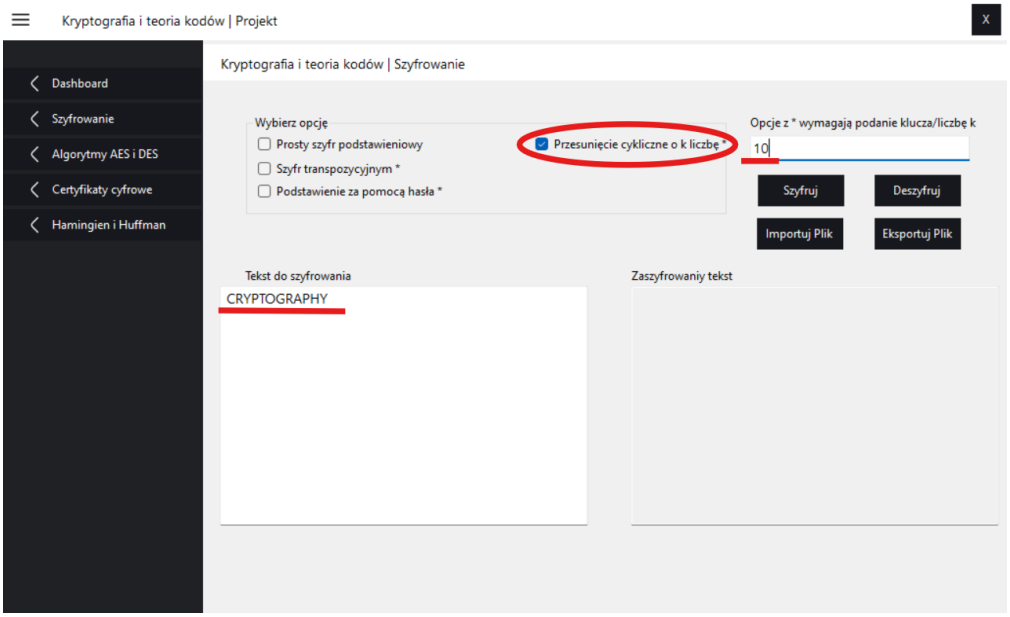
W aplikacji użytkownik ma możliwość wprowadzenia tekstu jawnego do zaszyfrowania oraz określenia wartości przesunięcia k, która może być liczbą całkowitą dodatnią lub ujemną. Po zaszyfrowaniu tekstu użytkownik może zobaczyć wygenerowany szyfrogram oraz odszyfrować go, podając tę samą wartość k.

3.4.3 Przykłady działania

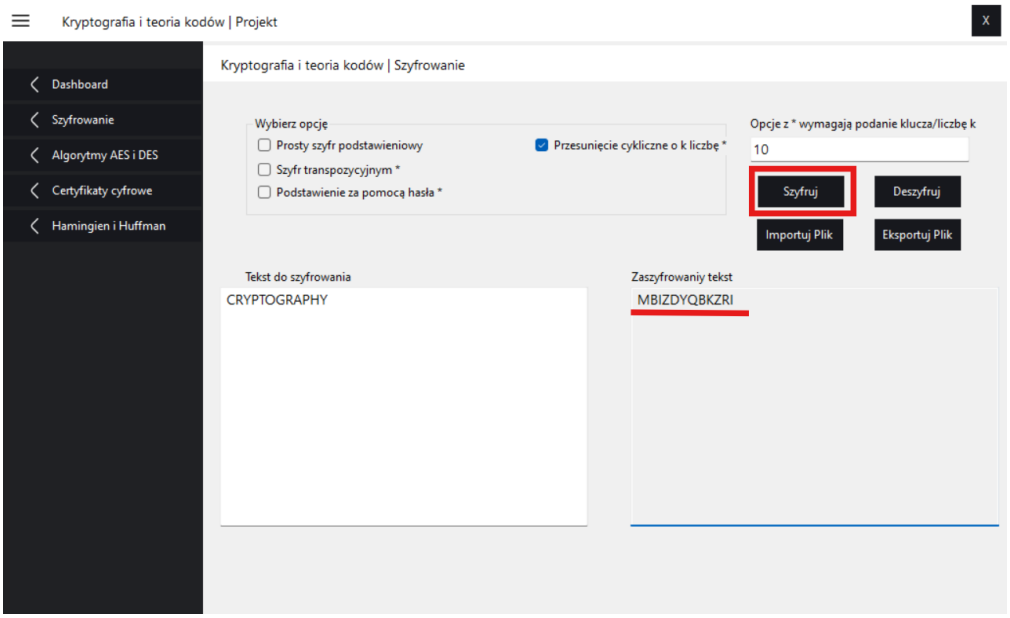
Tekst jawny	**Wartość k**	**Szyfrogram**
ABCD	3	DEFG
HELLO	5	MJQQT
WORLD	-3	TOLIA
CRYPTOGRAPHY	10	MBIZDYQBKZRI

Tabela 4: Przykłady szyfrowania z przesunięciem cyklicznym o k

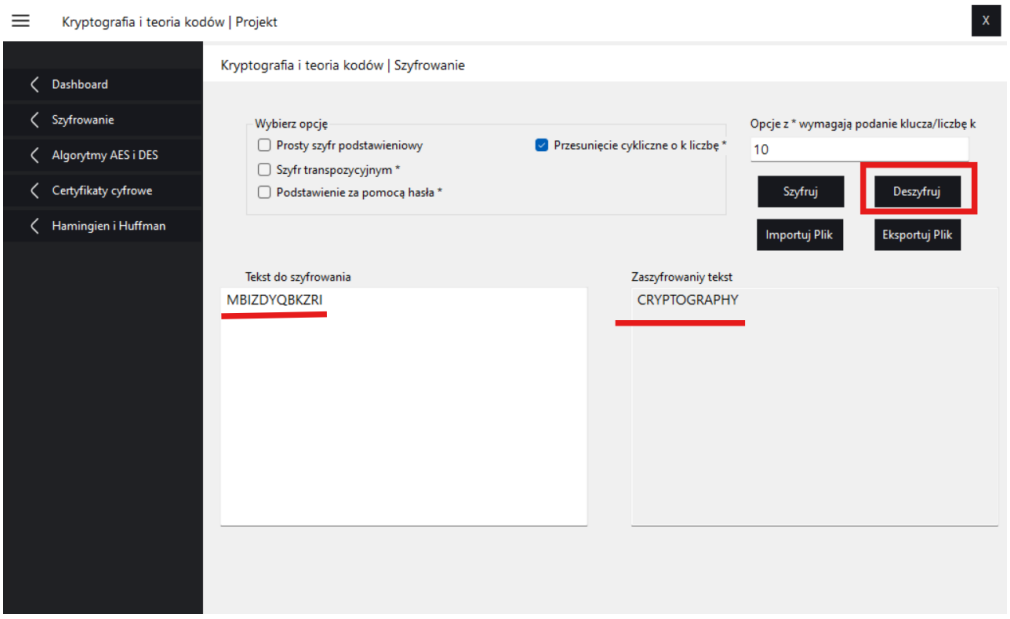
3.4.4 Przykładowe ekrany aplikacji



Rysunek 10: Pole wprowadzania tekstu jawnego i wartości przesunięcia k.



Rysunek 11: Wyświetlenie szyfrogramu po zastosowaniu przesunięcia cyklicznego.



Rysunek 12: Wyświetlenie szyfrogramu po wstecznym przesunięcia cyklicznego.

3.4.5 Wady i zalety

Szyfr z przesunięciem cyklicznym ma swoje zalety i wady. Do jego zalet należy bardzo prosta implementacja oraz przydatność w nauce podstaw kryptografii. Z drugiej strony, jego niski poziom bezpieczeństwa sprawia, że jest łatwy do złamania metodą analizy częstotliwości. Dodatkowo klucz, czyli wartość k, musi być znany obu stronom komunikacji.

4 Algorytmy współczesne

4.1 AES (Advanced Encryption Standard)

Advanced Encryption Standard (AES) jest jednym z najbardziej zaawansowanych i powszechnie stosowanych algorytmów szyfrowania symetrycznego. Został zaprojektowany przez Vincenta Rijmena i Joana Daemena, a w 2001 roku został ogłoszony standardem przez NIST (National Institute of Standards and Technology).

4.1.1 Opis algorytmu

AES działa na blokach danych o stałym rozmiarze 128 bitów i wspiera długości kluczy: 128 bitów (AES-128), 192 bity (AES-192) oraz 256 bitów (AES-256). W aplikacji zaimplementowano szyfrowanie i deszyfrowanie plików przy użyciu algorytmu AES. Klucz szyfrowania jest generowany na podstawie statycznego hasła: **Password2123**. Proces szyfrowania obejmuje kilka kroków, w tym załadowanie pliku wejściowego, wygenerowanie klucza szyfrowania na podstawie hasła oraz zaszyfrowanie danych w pliku, które są następnie zapisywane w nowym pliku wynikowym. Proces deszyfrowania działa analogicznie, wykorzystując ten sam klucz.

4.1.2 Funkcjonalność w aplikacji

W aplikacji użytkownik ma możliwość wyboru pliku do zaszyfrowania, automatycznego generowania szyfrogramu w pliku wynikowym oraz deszyfrowania zaszyfrowanego pliku, o ile używane jest hasło **Password2123**.

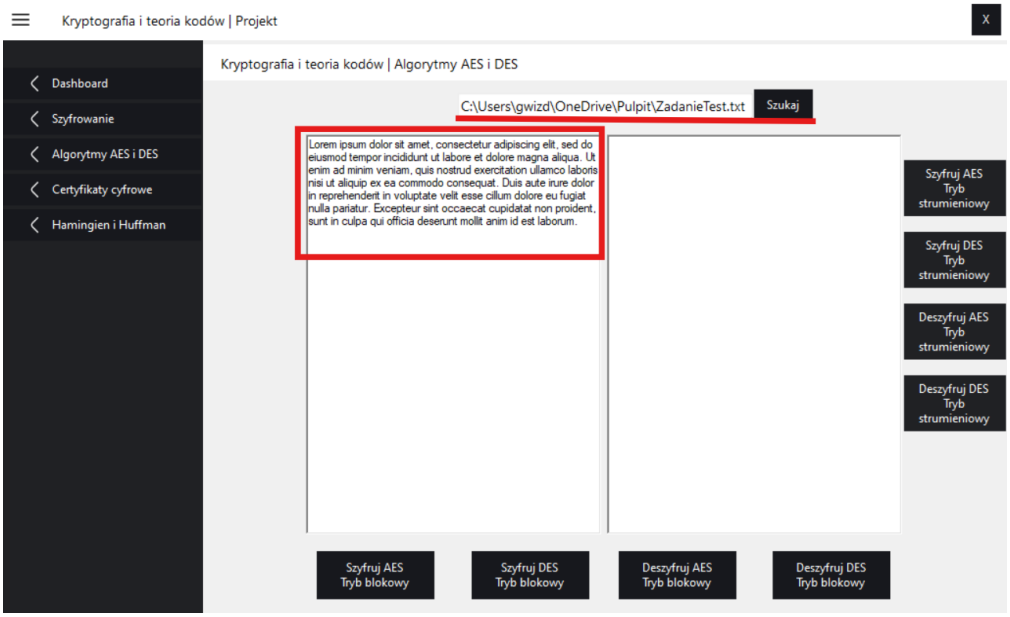
4.1.3 Przykłady działania

Przykłady działania aplikacji przy użyciu algorytmu AES znajdują się w poniższej tabeli.

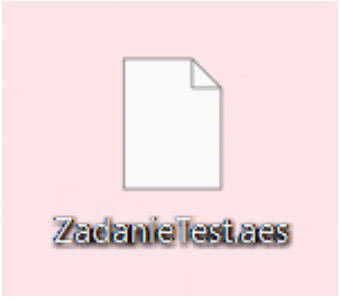
Nazwa pliku wejściowego	**Hasło szyfrowania**	**Nazwa pliku wynikowego**
ZadanieTest.txt	Password2123	ZadanieTest.aes
dokument.pdf	Password2123	dokument.aes

Tabela 5: Przykłady szyfrowania plików z użyciem algorytmu AES

4.1.4 Przykładowe ekrany aplikacji



Rysunek 13: Ekran wyboru pliku do szyfrowania za pomocą algorytmu AES.



Rysunek 14: Ekran wyświetlania wynikowego pliku szyfrogramu.

4.1.5 Zalety i wady algorytmu AES w aplikacji

Algorytm AES ma wiele zalet, takich jak możliwość szyfrowania plików o dowolnym formacie, bezpieczeństwo gwarantowane przez silny klucz szyfrowania oraz automatyczne generowanie szyfrogramu bez konieczności ręcznej obsługi klucza. Z drugiej strony, wykorzystanie statycznego hasła obniża bezpieczeństwo w przypadku jego ujawnienia, a pliki wynikowe wymagają użycia tego samego hasła do deszyfrowania.

4.2 DES (Data Encryption Standard)

Data Encryption Standard (DES) jest jednym z klasycznych algorytmów szyfrowania symetrycznego, opracowanym przez IBM w latach 70. XX wieku. Choć obecnie uznawany jest za mniej bezpieczny w stosunku do nowszych standardów (np. AES), nadal znajduje zastosowanie w celach edukacyjnych i w systemach o niższych wymaganiach bezpieczeństwa.

4.2.1 Opis algorytmu

DES operuje na blokach danych o wielkości 64 bitów i wykorzystuje klucz o długości 56 bitów. Proces szyfrowania obejmuje szereg permutacji i podstawień, realizowanych w 16 rundach, co zapewnia bezpieczeństwo kryptograficzne. W aplikacji zaimplementowano operacje szyfrowania i deszyfrowania plików z wykorzystaniem statycznego hasła: **Password2123**.

4.2.2 Proces szyfrowania i deszyfrowania

Działanie algorytmu w aplikacji obejmuje następujące kroki: użytkownik wybiera plik do zaszyfrowania, na podstawie hasła generowany jest klucz szyfrowania, dane z pliku wejściowego są szyfrowane i zapisywane w nowym pliku wynikowym. W przypadku deszyfrowania, dane szyfrogramu są odczytywane i odszyfrowywane za pomocą tego samego hasła.

4.2.3 Funkcjonalność w aplikacji

W aplikacji użytkownik może wybrać plik wejściowy do szyfrowania, generować pliki szyfrogramów oraz deszyfrować pliki szyfrogramów, korzystając z tego samego hasła (**Password2123**).

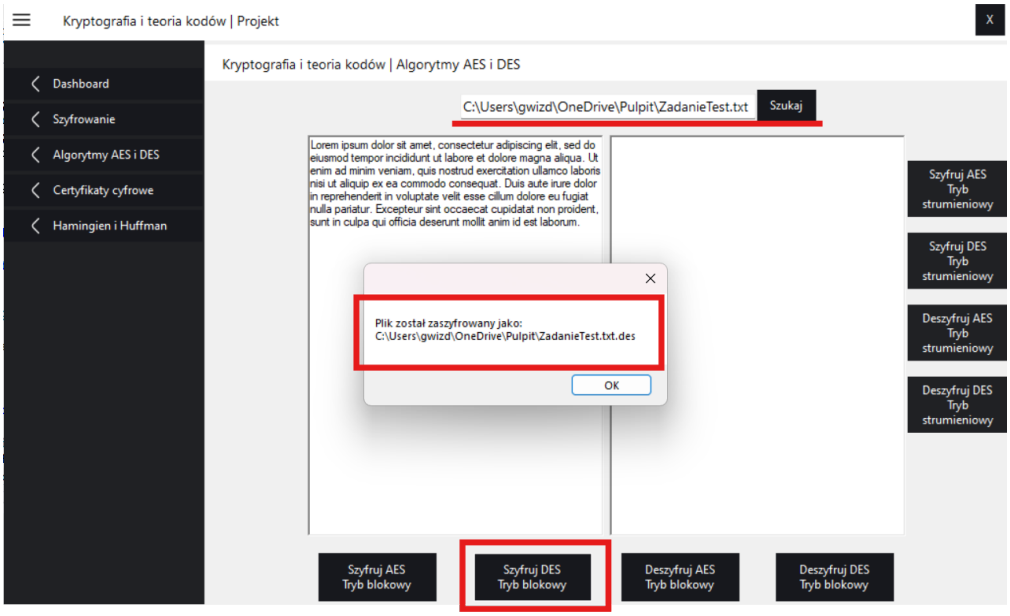
4.2.4 Przykłady działania

Poniżej przedstawiono przykładowe operacje szyfrowania i deszyfrowania w aplikacji:

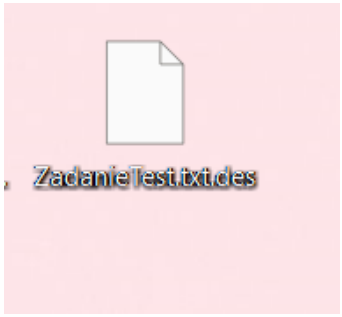
Nazwa pliku wejściowego	**Hasło szyfrowania**	**Nazwa pliku wynikowego**
ZadanieTest.txt	Password2123	ZadanieTest.txt.des
obraz.png	Password2123	obraz.png.des

Tabela 6: Przykłady szyfrowania plików z użyciem algorytmu DES

4.2.5 Przykładowe ekrany aplikacji



Rysunek 15: Ekran wyboru pliku do szyfrowania za pomocą algorytmu DES.



Rysunek 16: Ekran wyświetlania wyniku szyfrowania pliku.

4.2.6 Zalety i wady algorytmu DES w aplikacji

Algorytm DES ma swoje zalety, takie jak prosta implementacja, szczególnie w celach edukacyjnych oraz możliwość szyfrowania plików dowolnego formatu. Jednakże użycie statycznego hasła (**Password2123**) ogranicza bezpieczeństwo w przypadku jego ujawnienia, a DES jest podatny na ataki kryptograficzne i nie spełnia współczesnych standardów bezpieczeństwa.

5 Obsługa podpisu cyfrowego

Obsługa podpisu cyfrowego jest kluczową funkcjonalnością aplikacji, która pozwala na zapewnienie integralności i autentyczności danych. W tej sekcji omówiono procesy związane z generowaniem, weryfikacją podpisów cyfrowych oraz analizą informacji o certyfikacie używanym do weryfikacji podpisu.

5.1 Opis działania podpisu cyfrowego

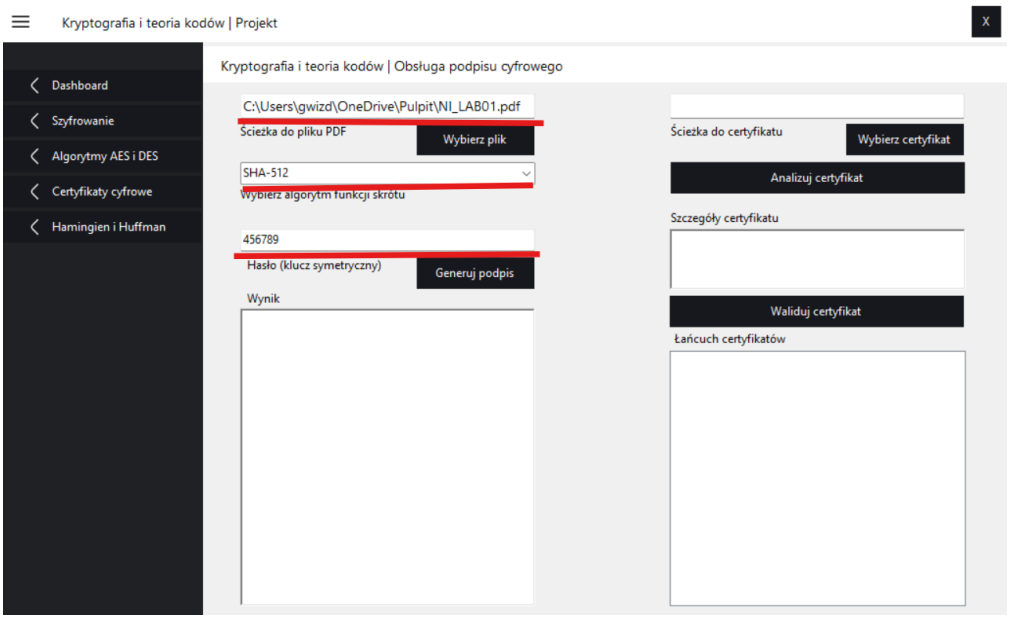
Podpis cyfrowy jest realizowany w oparciu o kryptografię asymetryczną. Klucz prywatny służy do generowania podpisu, natomiast klucz publiczny – do jego weryfikacji. Proces w aplikacji przebiega następująco: użytkownik wybiera plik, który chce podpisać, a aplikacja generuje podpis cyfrowy za pomocą klucza prywatnego użytkownika. Następnie użytkownik wczytuje podpisany plik oraz odpowiedni certyfikat, po czym aplikacja weryfikuje autentyczność podpisu za pomocą klucza publicznego zawartego w certyfikacie. Dodatkowo aplikacja umożliwia wyświetlenie szczegółowych informacji o certyfikacie używanym do weryfikacji podpisu.

5.2 Proces obsługi podpisu cyfrowego w aplikacji

Poniżej przedstawiono szczegółowe etapy pracy z podpisem cyfrowym w aplikacji.

5.2.1 Generowanie podpisu

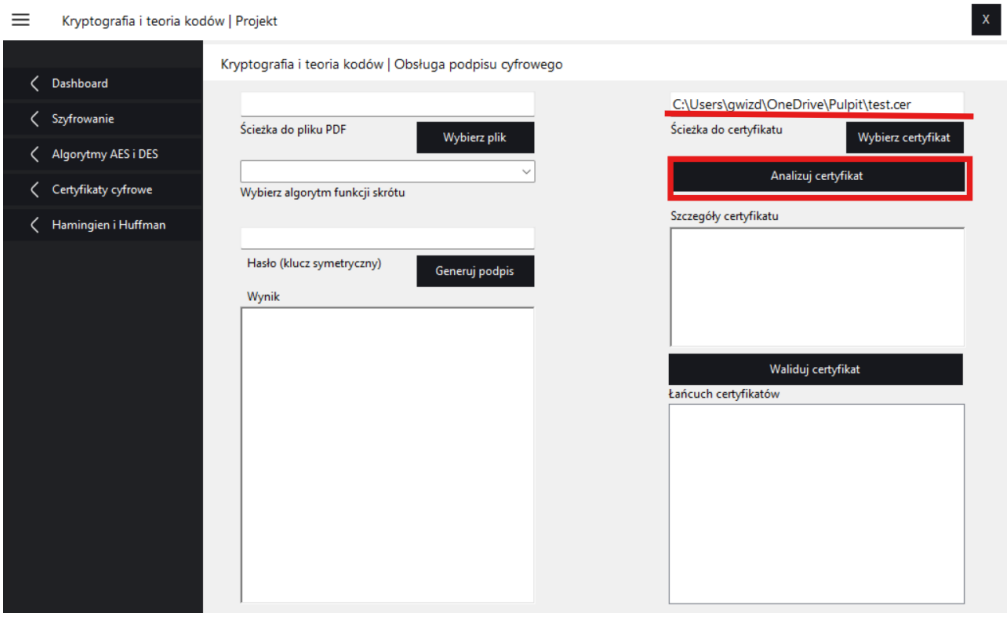
Aby wygenerować podpis, użytkownik wybiera plik do podpisania, następnie aplikacja prosi o wskazanie klucza prywatnego (np. pliku w formacie `.pem`). Podpis cyfrowy jest generowany i zapisywany w osobnym pliku (np. `plik_podpis.sig`).



Rysunek 17: Ekran generowania podpisu cyfrowego.

5.2.2 Weryfikacja podpisu

W procesie weryfikacji użytkownik wczytuje podpisany plik oraz plik podpisu (`plik_podpis.sig`), a następnie wybiera certyfikat zawierający klucz publiczny (`certyfikat.cer`). Aplikacja sprawdza, czy podpis jest zgodny z danymi w pliku i wyświetla wynik weryfikacji.



Rysunek 18: Ekran weryfikacji podpisu cyfrowego.

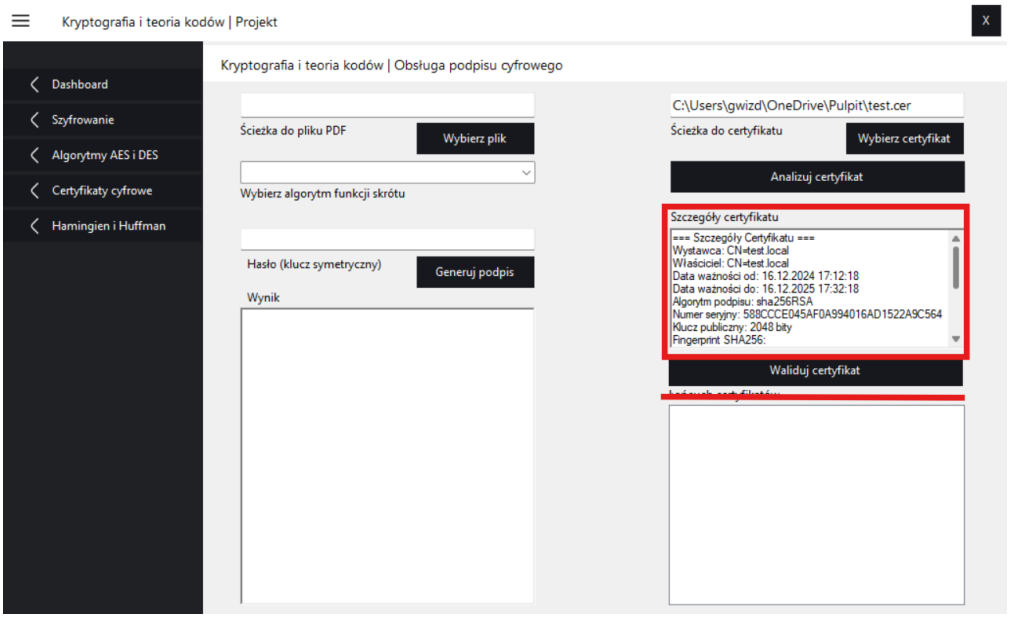
5.2.3 Analiza informacji o certyfikacie

Aplikacja oferuje możliwość wyświetlenia szczegółowych informacji o certyfikacie używanym do weryfikacji podpisu. Przykładowe dane widoczne w aplikacji to: - **Nazwa wystawcy:** Organizacja, która wydała certyfikat. - **Nazwa właściciela:** Tożsamość właściciela certyfikatu. s- **Okres ważności:** Data początkowa i końcowa obowiązywania certyfikatu. - **Numer seryjny:** Unikalny identyfikator certyfikatu. - **Algorytm podpisu:** Algorytm używany do podpisania certyfikatu (np. `SHA-256`).

Przykładowa tabela z informacjami o certyfikacie:

Parametr	**Wartość**
Nazwa wystawcy	Let's Encrypt
Nazwa właściciela	Jan Kowalski
Okres ważności	01.01.2024 – 31.12.2024
Numer seryjny	1234-5678-9012-3456
Algorytm podpisu	SHA-256

Tabela 7: Przykładowe informacje o certyfikacie.



Rysunek 19: Ekran analizy certyfikatu.

5.3 Zalety i wady obsługi podpisu cyfrowego w aplikacji

Obsługa podpisu cyfrowego ma swoje zalety i wady. Do zalet należy intuicyjny interfejs użytkownika, szczegółowa analiza informacji o certyfikacie oraz obsługa plików podpisu w formacie `.sig`. Z drugiej strony, wadami są brak opcji dynamicznej zmiany algorytmu podpisu oraz brak wsparcia dla wielu kluczy jednocześnie.

5.4 Podsumowanie

Funkcja obsługi podpisu cyfrowego w aplikacji pozwala na generowanie, weryfikację i analizę podpisów cyfrowych. Szczególną uwagę zwrócono na łatwość obsługi oraz szczegółową prezentację informacji o certyfikacie, co czyni aplikację użytecznym narzędziem zarówno w praktycznych zastosowaniach, jak i celach edukacyjnych.

6 Algorytmy kodowania

6.1 Kodowanie Hamming

Kodowanie Hamming to metoda wykrywania i korekcji błędów, która umożliwia dodanie bitów kontrolnych do danych w celu wykrywania i poprawiania pojedynczych błędów w transmisji. W tej sekcji omówiono sposób działania algorytmu, jego implementację w aplikacji oraz sposób wizualizacji wyników kodowania i dekodowania.

6.1.1 Opis działania algorytmu Hamming

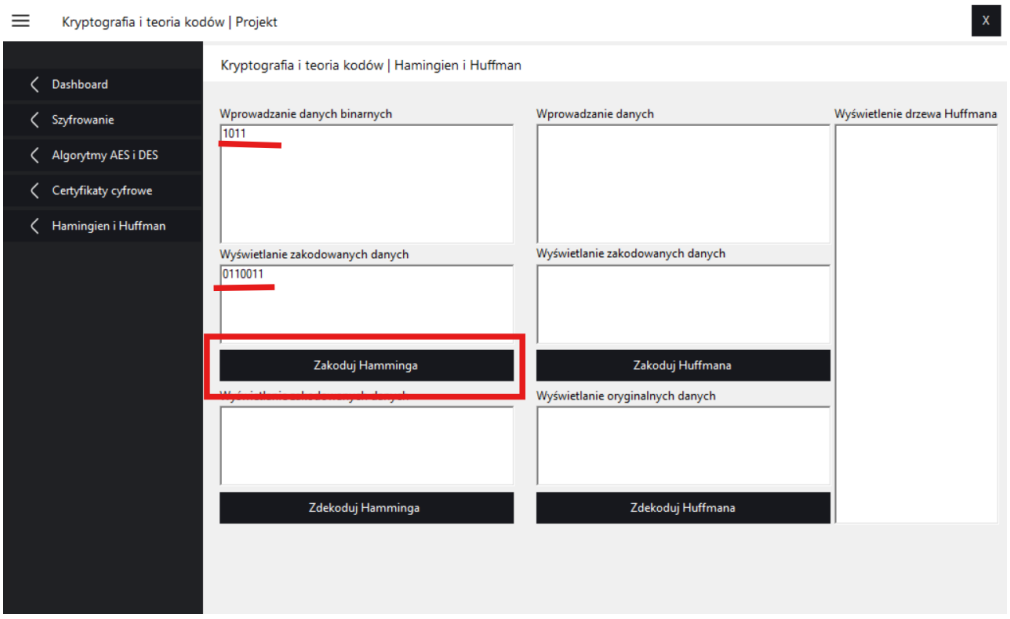
Algorytm Hamming działa na podstawie następujących kroków:

1. Wprowadzenie bitów kontrolnych:
    - Wstawienie bitów parzystości na odpowiednich pozycjach w danych.
    - Liczba bitów kontrolnych zależy od długości danych (np. dla 4-bitowych danych potrzeba 3 bitów kontrolnych).
2. Obliczenie wartości bitów kontrolnych:
    - Każdy bit kontrolny sprawdza parzystość bitów w ustalonych grupach.
    - Algorytm opiera się na zasadzie sum parzystych.
3. Dekodowanie i korekcja błędów:
    - Weryfikacja parzystości bitów.
    - Identyfikacja miejsca błędu (jeśli występuje).
    - Korekcja błędu poprzez zmianę odpowiedniego bitu.

6.1.2 Proces kodowania i dekodowania w aplikacji

Kodowanie danych:

1. Użytkownik wprowadza dane wejściowe w postaci binarnej (np. `1011`).
2. Aplikacja generuje zakodowane dane, wstawiając odpowiednie bity kontrolne (np. `1011010`).



Rysunek 20: Ekran kodowania danych za pomocą algorytmu Hamming.

Dekodowanie i korekcja błędów:

1. Użytkownik wprowadza dane zakodowane (np. `1011010`).
2. Aplikacja analizuje bity kontrolne w celu wykrycia błędu.
3. Jeśli wystąpił pojedynczy błąd, aplikacja identyfikuje jego lokalizację i dokonuje korekcji.
4. Wyświetlane są zarówno dane poprawione, jak i oryginalne.

6.1.3 Tabela wyników kodowania

Przykładowa tabela prezentująca kodowanie i dekodowanie danych przy użyciu algorytmu Hamming:

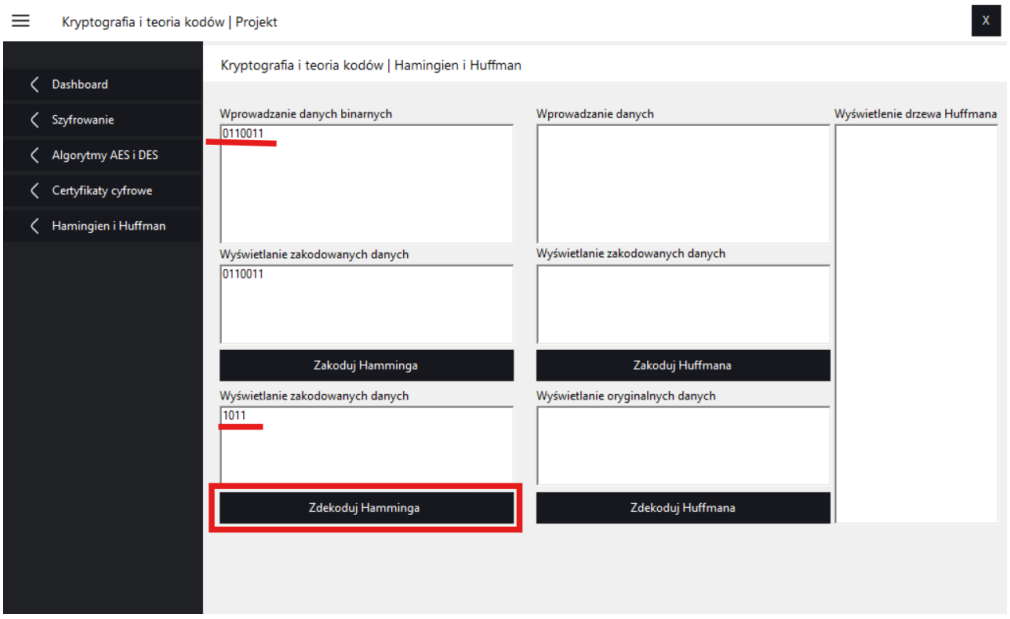
Dane wejściowe	Dane zakodowane	Dane z błędem	Dane poprawione
1011	1011010	1011000	1011010
1101	1101110	1100110	1101110

Tabela 8: Przykład kodowania i korekcji błędów.

6.1.4 Zalety i ograniczenia algorytmu Hamming

Zalety:

- Skuteczna korekcja pojedynczych błędów w danych.
- Prosta implementacja w aplikacjach edukacyjnych i technicznych.



Rysunek 21: Ekran dekodowania i korekcji błędów za pomocą algorytmu Hamming.

Ograniczenia:

- Algorytm nie obsługuje korekcji wielu błędów jednocześnie.
- Wydajność zmniejsza się dla dużych zestawów danych.

6.1.5 Podsumowanie

Kodowanie Hamming w aplikacji zostało zaprojektowane jako intuicyjne narzędzie do nauki i analizy algorytmu. Użytkownik może kodować, dekodować oraz analizować błędy w danych, co czyni aplikację praktycznym narzędziem zarówno dla studentów, jak i specjalistów. Szczegółowe wyniki oraz intuicyjne wizualizacje poprawiają zrozumienie działania algorytmu.

6.2 Kodowanie Huffman

Kodowanie Huffman to jedna z najpopularniejszych metod kompresji danych, której celem jest minimalizacja rozmiaru danych przy zachowaniu ich pełnej informacji. Algorytm ten opiera się na przypisaniu krótszych kodów binarnych do symboli występujących w danych częściej, a dłuższych kodów do rzadziej występujących symboli. W tej sekcji omówiono działanie algorytmu Huffmana, jego implementację w aplikacji oraz wyniki kompresji i dekompresji danych.

6.2.1 Opis działania algorytmu Huffman

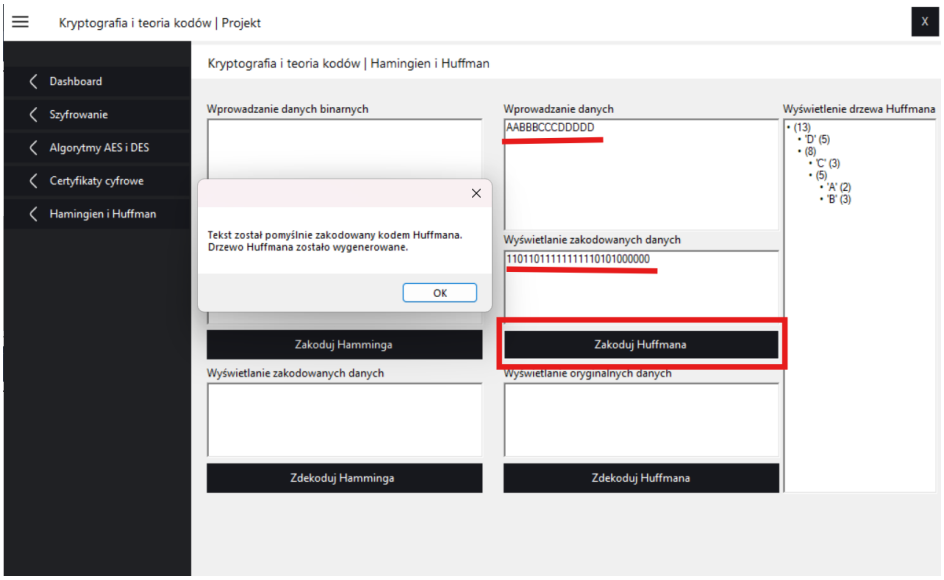
Algorytm Huffmana działa na podstawie następujących kroków:

1. Analiza częstotliwości symboli w danych:
    - Na początku algorytm zlicza, jak często każdy symbol (np. litery, liczby, znaki) występuje w danych wejściowych.
2. Budowanie drzewa Huffmana:
    - Na podstawie częstotliwości tworzy się drzewo binarne, w którym symbole o najniższej częstotliwości są łączone w gałęzie.
3. Przydzielanie kodów binarnych:
    - Po zbudowaniu drzewa, przypisuje się kody binarne do poszczególnych symboli. Symbole o większej częstotliwości otrzymują krótsze kody.
4. Kompresja danych:
    - Dane są kompresowane, zamieniając każdy symbol na odpowiadający mu kod binarny.
5. Dekompresja danych:
    - Aby odzyskać oryginalne dane, należy przeanalizować kod binarny, zaczynając od korzenia drzewa Huffmana.

6.2.2 Proces kompresji i dekompresji w aplikacji

Kompresja danych:

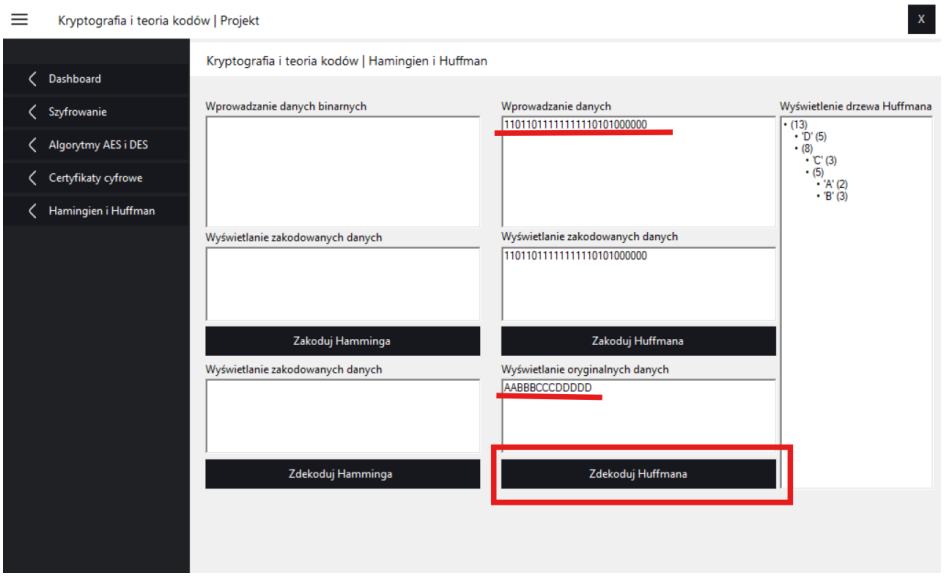
1. Użytkownik wprowadza dane wejściowe, na przykład tekst (np. "AABBBCCCDDDDD").
2. Aplikacja analizuje częstotliwości wystąpień symboli.
3. Następnie generuje drzewo Huffmana, przydziela kody binarne i kompresuje dane.
4. Zakończenie kompresji prowadzi do wyświetlenia zakodowanego ciągu bitów.



Rysunek 22: Ekran kompresji danych za pomocą algorytmu Huffman.

Dekompresja danych:

1. Użytkownik wprowadza zakodowane dane binarne.
2. Aplikacja analizuje kod binarny za pomocą drzewa Huffmana i dekompresuje go do postaci oryginalnych danych.
3. Wyświetlane są zarówno dane skompresowane, jak i dekompresowane.



Rysunek 23: Ekran dekompresji danych za pomocą algorytmu Huffman.

6.2.3 Tabela wyników kompresji

Przykładowa tabela prezentująca dane wejściowe, kodowanie Huffman oraz stopień kompresji:

Dane wejściowe	Zakodowane dane	Stopień kompresji
AABBBCCCDDDDD	11001011100111101101100	65%
ABCDABCDABCD	10100111010101100101100	60%

Tabela 9: Przykład kompresji danych za pomocą algorytmu Huffman.

6.2.4 Zalety i ograniczenia algorytmu Hamming

Zalety:

- Bardzo efektywne w przypadku danych o wyraźnie różnej częstotliwości występowania symboli.
- Zapewnia optymalną kompresję w porównaniu do innych algorytmów bezstratnych.
- Prosta implementacja w aplikacjach kompresyjnych.

Ograniczenia:

- Algorytm Huffmana może nie działać optymalnie w przypadku mało zróżnicowanych danych (np. tekstów z równą liczbą wystąpień symboli).
- Wymaga przechowywania drzewa Huffmana, co może zwiększać rozmiar skompresowanych danych.

6.2.5 Podsumowanie

Algorytm Huffmana w aplikacji jest skutecznym narzędziem do kompresji danych, szczególnie w przypadku dużych zbiorów danych tekstowych o zróżnicowanej częstotliwości występowania symboli. Dzięki prostocie implementacji i efektywności kompresji stanowi doskonałe rozwiązanie do zastosowań, w których ważna jest minimalizacja rozmiaru plików przy zachowaniu ich pełnej integralności. Aplikacja umożliwia łatwe kompresowanie i dekompresowanie danych, a także oferuje wizualizacje procesu kompresji i dekompresji, co może być szczególnie pomocne w kontekście nauki o algorytmach kompresji.

7 Podsumowanie

W niniejszej dokumentacji zaprezentowano implementację i zastosowanie różnych algorytmów kryptograficznych oraz kompresyjnych w ramach stworzonej aplikacji. Każdy z zaprezentowanych algorytmów ma swoje specyficzne właściwości, które decydują o ich zastosowaniach i skuteczności w różnych kontekstach.

W sekcji dotyczącej szyfrów omówiono zarówno klasyczne algorytmy, takie jak szyfr podstawieniowy i szyfr transpozycyjny, jak i bardziej zaawansowane metody, takie jak szyfr AES i DES, które zostały zaimplementowane na plikach z użyciem statycznego hasła "Password2123". W przypadku algorytmów AES i DES szczególną uwagę poświęcono ich implementacji na plikach, co pozwoliło na analizę efektywności kompresji oraz dekompresji danych.

Kolejnym omawianym zagadnieniem była obsługa podpisu cyfrowego, w której użytkownicy mogą weryfikować autentyczność i integralność danych. Szczegółowo przedstawiono sposób generowania certyfikatów oraz analizę związanych z nimi informacji, co umożliwia pełną weryfikację podpisu.

W kontekście kompresji danych zaprezentowano dwa kluczowe algorytmy: kodowanie Hamming i kodowanie Huffmana. Algorytmy te pozwalają na efektywne zmniejszenie rozmiaru danych bez utraty informacji. W aplikacji użytkownicy mogą w łatwy sposób kompresować i dekompresować dane, co zostało zilustrowane przy pomocy przykładów.

Wszystkie opisane algorytmy są zintegrowane w ramach jednej aplikacji, co umożliwia użytkownikowi korzystanie z różnych metod kryptograficznych i kompresyjnych w zależności od jego potrzeb. Aplikacja została zaprezentowana za pomocą interfejsu graficznego, który umożliwia łatwą interakcję z użytkownikiem oraz prezentację wyników w przejrzysty sposób.

Podsumowując, zaprezentowane algorytmy i techniki stanowią solidną podstawę do realizacji aplikacji zapewniającej bezpieczeństwo danych oraz ich efektywną kompresję. Ich implementacja, zarówno z perspektywy teorii, jak i praktyki, stanowi cenne narzędzie w różnych dziedzinach informatyki, takich jak bezpieczeństwo cyfrowe, przechowywanie danych oraz optymalizacja rozmiaru plików.

Dzięki zastosowaniu nowoczesnych algorytmów kryptograficznych oraz skutecznych metod kompresji, aplikacja jest w stanie sprostać wymaganiom współczesnych użytkowników. Umożliwia ona nie tylko ochronę danych przed nieautoryzowanym dostępem, ale także oszczędność miejsca na dysku poprzez efektywne zarządzanie rozmiarem plików. W przyszłości planowane jest rozszerzenie funkcjonalności aplikacji o dodatkowe algorytmy oraz możliwość integracji z innymi systemami informatycznymi.