Akademia Nauk Stosowanych w Nowym Sączu **Kryptografia i teoria kodów - projekt**	
Nazwisko i imię: **Zwolenik Arkadiusz**	Ocena projektu:
Data:	Grupa: **P3**

1. Czym jest kryptografia?

Kryptografia to dziedzina nauki zajmująca się tworzeniem metod zabezpieczania informacji oraz zapewniania jej poufności, integralności i autentyczności. Głównym celem kryptografii jest umożliwienie bezpiecznej komunikacji oraz ochrony danych przed nieautoryzowanym dostępem. Termin pochodzi od greckich słów kryptos (ukryty) i graphein (pisać), co dosłownie oznacza „ukryte pisanie”.

1.1 Podstawowe cele kryptografii

Kryptografia dąży do spełnienia kilku kluczowych zasad bezpieczeństwa informacji, takich jak:

- **Poufność** – tylko uprawnione osoby mogą uzyskać dostęp do informacji.
- **Integralność** – dane nie mogą być zmieniane lub modyfikowane przez osoby nieuprawnione.
- **Autentyczność** – możliwość potwierdzenia tożsamości nadawcy wiadomości lub źródła danych.
- **Niezaprzeczalność** – nadawca nie może zaprzeczyć wysłaniu wiadomości, co jest istotne np. w podpisach cyfrowych.

1.2 Podstawowe typy kryptografii

- **Kryptografia symetryczna** (jednokluczowa):

  → Wykorzystuje jeden klucz do szyfrowania i deszyfrowania danych, co oznacza, że nadawca i odbiorca muszą posiadać ten sam klucz.

  → Jest bardzo wydajna i stosowana m.in. w algorytmach takich jak **DES**, czy **AES**.

  → Wadą jest konieczność bezpiecznej wymiany klucza między stronami.

- **Kryptografia asymetryczna** (dwukluczowa):

  → Wykorzystuje dwa różne, ale matematycznie powiązane klucze: **klucz publiczny** (do szyfrowania) i **klucz prywatny** (do deszyfrowania).

  → Klucz publiczny może być udostępniony każdemu, natomiast klucz prywatny jest trzymany w tajemnicy przez właściciela.

  → Używana m.in. w algorytmach **RSA**, **ECC** (krzywych eliptycznych).

  → Jest mniej wydajna od kryptografii symetrycznej, ale nie wymaga bezpiecznej wymiany kluczy.

- **Kryptografia hybrydowa**:

  → Łączy kryptografię symetryczną i asymetryczną, wykorzystując szyfrowanie asymetryczne do bezpiecznej wymiany klucza symetrycznego, a następnie symetryczne do szyfrowania danych.

  → Stosowana np. w protokołach SSL/TLS do bezpiecznych połączeń internetowych.

1.3 Podstawowe techniki kryptograficzne

1) **Szyfrowanie** – proces przekształcania tekstu jawnego (czytelnego) w zaszyfrowany, czyli tekst nieczytelny dla osób nieupoważnionych.

2) **Deszyfrowanie** – proces przywracania zaszyfrowanego tekstu do jego pierwotnej, czytelnej formy.

3) **Haszowanie** – przekształcenie danych dowolnej długości w stałej długości skrót (hash), co pozwala sprawdzić integralność danych (algorytmy haszujące to m.in. SHA-256, MD5).

4) **Podpis cyfrowy** – technika kryptograficzna, która umożliwia potwierdzenie autentyczności i integralności dokumentu lub wiadomości.

1.4 Przykłady zastosowań kryptografii

Kryptografia jest szeroko stosowana w różnych dziedzinach życia codziennego, takich jak:

- **Bankowość internetowa** i transakcje online (zapewnienie bezpieczeństwa przelewów).
- **Komunikacja internetowa** (np. szyfrowane wiadomości w aplikacjach typu WhatsApp czy Signal).
- **Podpisy cyfrowe** stosowane w e-dokumentach i komunikacji e-mail.
- **Bezpieczeństwo Wi-Fi** (protokoły WPA, WPA2, WPA3 do zabezpieczania sieci bezprzewodowych).
- **Autoryzacja i uwierzytelnianie** użytkowników w serwisach internetowych i aplikacjach.

2. Cel i zakres pracy

Celem pracy jest stworzenie prostej aplikacji webowej w technologii Blazor WebAssembly umożliwiającej użytkownikom przetestowanie różnych algorytmów szyfrowania. Aplikacja będzie dostarczać intuicyjny interfejs, pozwalający na wpisywanie danych, wybór kluczy oraz – w przypadku niektórych metod – możliwość przesłania pliku do zaszyfrowania. Narzędzie ma na celu wprowadzenie użytkowników do zagadnień kryptografii, demonstrując różne podejścia do szyfrowania i deszyfrowania danych.

Aplikacja pozwoli użytkownikowi wybrać spośród dostępnych algorytmów szyfrowania, zaszyfrować i odszyfrować dane przy użyciu wybranego algorytmu oraz zrozumieć podstawowe różnice między poszczególnymi typami szyfrowania.

Zakres pracy obejmuje:

1) Opracowanie struktury aplikacji w **Blazor WebAssembly**:

    a) Stworzenie prostego interfejsu użytkownika z **menu**, które pozwala wybrać jeden z dostępnych algorytmów szyfrowania.

    b) Implementacja formularzy umożliwiających **wprowadzanie danych**, **podanie klucza szyfrującego** lub **załadowanie pliku** do szyfrowania.

2) Implementacja podstawowych algorytmów szyfrowania:

    a) **Szyfry klasyczne**: np. szyfr polialfabetyczny (Vigenère) i przestawieniowy.

    b) **Szyfrowanie transpozycyjne**: proste algorytmy, które zmieniają pozycję znaków.

    c) **Szyfrowanie symetryczne**: implementacja podstawowych algorytmów szyfrowania blokowego (DES i AES) oraz strumieniowego.

3) Interaktywne **testowanie** algorytmów szyfrowania:

    a) Każdy moduł umożliwi **wpisanie danych lub wczytanie pliku**, podanie klucza szyfrującego oraz wykonanie operacji szyfrowania i deszyfrowania.

    b) Wyświetlanie **wyników** szyfrowania w czytelnej formie, z możliwością skopiowania.

4) Weryfikacja poprawności działania algorytmów, czyli testowanie algorytmów w celu sprawdzenia poprawności procesu szyfrowania i deszyfrowania, aby wyniki były zgodne z teoretycznymi wzorcami.

5) Dokumentacja projektu i wnioski, czyli opis struktury i funkcji aplikacji, wyjaśnienie działania poszczególnych algorytmów, opis interfejsu użytkownika oraz przedstawienie obszarów do dalszego rozwoju aplikacji.

3. Aplikacja

3.1 UI

Poniższy zrzut ekranu przedstawia fragment menu głównego aplikacji, umożliwiający wybór spośród różnych typów algorytmów szyfrowania.

Rysunek 1. Fragment nawigacji pomiędzy różnymi typami szyfrowania

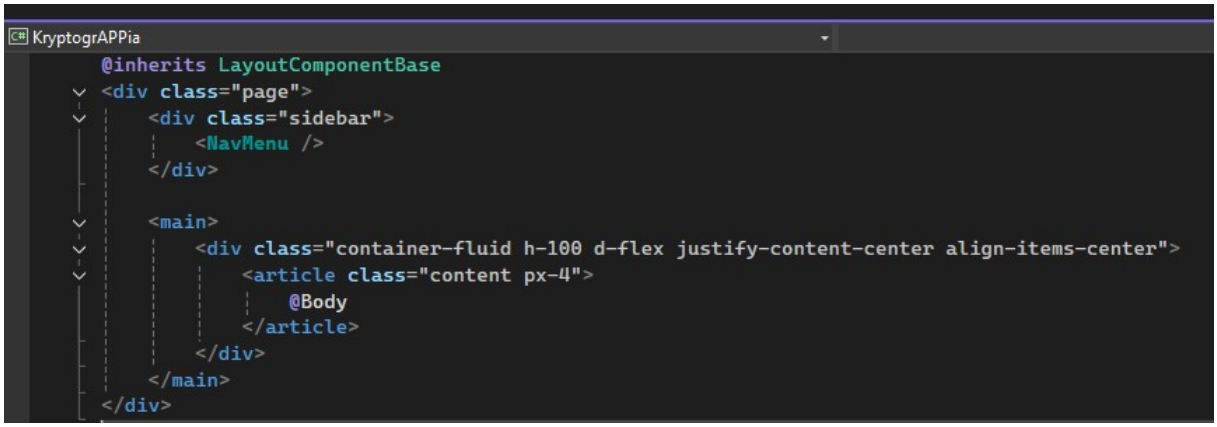
Poniższy zrzut ekranu przedstawia fragment menu głównego aplikacji, umożliwiający wybór spośród różnych typów algorytmów szyfrowania.

```
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">KryptogrAPPia</a>
        <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span aria-hidden="true"></span> System Polifalfabetyczny
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="transpozycja">
                <span aria-hidden="true"></span> System Transpozycyjny
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="blokowe_AES">
                <span aria-hidden="true"></span> Szyfrowanie Blokowe - AES
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="blokowe_DES">
                <span aria-hidden="true"></span> Szyfrowanie Blokowe - DES
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="strumieniowe">
                <span aria-hidden="true"></span> Szyfrowanie Strumieniowe
            </NavLink>
        </div>
    </nav>
</div>
```

Rysunek 2. Zrzut ekranu przedstawiający implementację menu nawigacji

Poniższe zrzuty ekranu przedstawiają implementację głównego layoutu aplikacji, który ułatwia tworzenie funkcjonalności poprzez eliminację potrzeby wielokrotnego powtarzania kodu odpowiedzialnego za ustawianie elementów HTML. Dzięki zastosowanemu układowi, wszelkie zmiany w wyglądzie layoutu można wprowadzać w jednym miejscu, co sprawia, że nie ma potrzeby modyfikowania układu w każdej kategorii szyfrowania z osobna. Takie podejście zapewnia większą spójność i ułatwia zarządzanie interfejsem aplikacji.

Rysunek 3. Strona główna aplikacji



Rysunek 4. Implementacja głównego layoutu aplikacji

3.2 Szyfr polialfabetyczny

Szyfr polialfabetyczny to rodzaj algorytmu szyfrowania, w którym do zamiany liter w wiadomości stosuje się różne zestawy znaków (alfabety), w zależności od pozycji liter w tekście. Celem tego podejścia jest zwiększenie bezpieczeństwa szyfrowania poprzez wprowadzenie większej zmienności w przypisaniu liter, co utrudnia złamanie szyfru przy użyciu klasycznych metod, takich jak analiza częstotliwości.

Najbardziej znanym przykładem szyfru polialfabetycznego jest szyfr Vigenère'a, w którym do szyfrowania wykorzystuje się klucz – słowo lub frazę, która określa, które litery alfabetu będą stosowane do każdej litery w tekście. Klucz jest powtarzany, aby dopasować jego długość do długości wiadomości, a każda litera w wiadomości jest szyfrowana przy użyciu odpowiedniej litery klucza.

W szyfrze Vigenère'a dla każdej litery tekstu jawnego wybierany jest inny alfabet (lub przesunięcie) zgodnie z literą w kluczu. Dzięki temu każda litera tekstu jest szyfrowana innym przesunięciem, co znacząco utrudnia analizę częstotliwościową, która jest skuteczną metodą łamania prostych szyfrów monoalfabetycznych, takich jak szyfr Cezara.

Szyfr Gronsfelda to wariant szyfru polialfabetycznego, w którym do szyfrowania tekstu wykorzystuje się liczby zamiast liter. Jest to metoda, która działa podobnie do szyfru Vigenère'a, ale klucz składa się z cyfr, a każda cyfra w kluczu określa liczbę przesunięcia dla odpowiadającej litery w tekście.

Zasada działania szyfru Gronsfelda:

- Klucz składa się z cyfr (np. 3 1 4 1), które określają liczbę przesunięcia dla liter w tekście.
- Każda cyfra w kluczu odpowiada liczbom przesunięć, które są stosowane do liter w tekście jawnym.
- Klucz jest powtarzany w razie potrzeby, aby dopasować go do długości tekstu.
- Aby zaszyfrować tekst, każda litera w wiadomości jest przesuwana o liczbę pozycji w alfabecie równą cyfrze z klucza. Jeśli przesunięcie przekroczy literę Z, zaczyna się od początku alfabetu.

Przykład:

Tekst: Ala ma kota

Klucz: 2137

Szyfrowanie:

1) A (przesunięcie o 2) → C
2) L (przesunięcie o 1) → M
3) A (przesunięcie o 3) → D
4) M (przesunięcie o 7) → T
5) A (przesunięcie o 2) → C
6) K (przesunięcie o 1) → L
7) O (przesunięcie o 3) → R
8) T (przesunięcie o 7) → A
9) A (przesunięcie o 2) → C

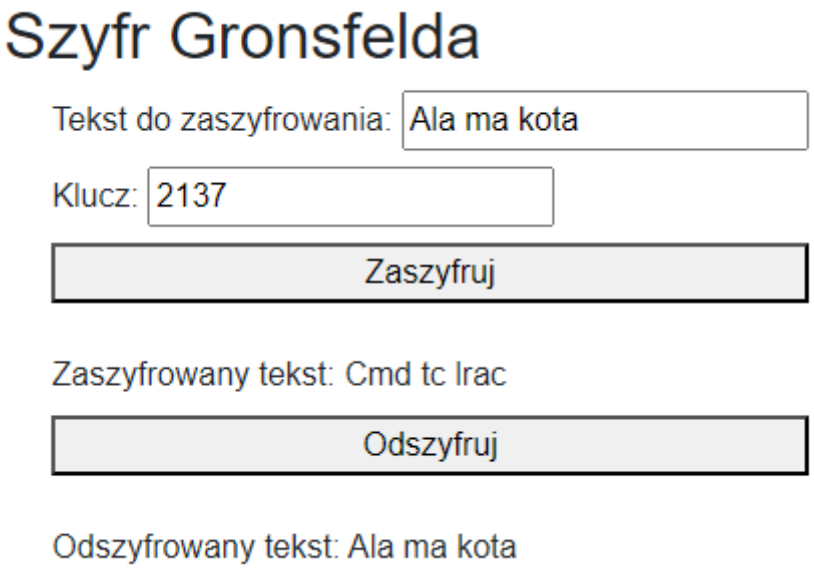
Wynik: "Cmd t clrac"

Funkcja GronsfeldCipher implementuje szyfr Gronsfelda, który jest wariantem szyfru Cezara, ale zamiast stałego przesunięcia używa klucza, składającego się z cyfr. Działa na tekście, przesuwając litery o wartość odpowiadającą cyfrom w kluczu. Funkcja przyjmuje trzy argumenty: tekst do zaszyfrowania lub odszyfrowania (text), klucz szyfrujący (cipherKey) i wartość logiczną (encrypt), która określa, czy funkcja ma szyfrować (gdy encrypt jest true) czy odszyfrowywać (gdy encrypt jest false). Funkcja iteruje przez każdy znak w tekście i jeśli jest literą, oblicza przesunięcie na podstawie cyfry w kluczu, cyklicznie przechodząc przez jego cyfry. Jeśli odszyfrowuje, przesunięcie jest obliczane w przeciwnym kierunku, czyli odejmuje wartość klucza od 26. Małe i duże litery są przesuwane w obrębie odpowiedniego alfabetu, a znaki

nieliterowe (np. spacje, przecinki) pozostają niezmienione. Na końcu funkcja zwraca zaszyfrowany lub odszyfrowany tekst, w zależności od wartości argumentu encrypt.

```
//Implementacja szyfru Gronsfelda
2 references
private string GronsfeldCipher(string text, string cipherKey, bool encrypt)
{
    var result = new StringBuilder();
    var keyIndex = 0;
    var keyLength = cipherKey.Length;

    foreach (var ch in text)
    {
        //Przesuwanie tylko liter
        if (char.IsLetter(ch))
        {
            //Określenie przesunięcia na podstawie cyfry w kluczu
            int shift = int.Parse(cipherKey[keyIndex % keyLength].ToString());

            //Jeśli odszyfrowujemy, to używamy przesunięcia w przeciwną stronę
            if (!encrypt)
            {
                shift = 26 - shift; //Odejmujemy przesunięcie zamiast dodawać
            }

            //Przesunięcie dla małych liter
            if (char.IsLower(ch))
            {
                result.Append((char)((((ch - 'a') + shift) % 26) + 'a'));
            }
            //Przesunięcie dla dużych liter
            else if (char.IsUpper(ch))
            {
                result.Append((char)((((ch - 'A') + shift) % 26) + 'A'));
            }

            keyIndex++; //Przechodzimy do kolejnej cyfry w kluczu
        }
        else
        {
            result.Append(ch); //Dodajemy inne znaki bez zmian (np. spacje, przecinki)
        }
    }

    return result.ToString();
}
```

Rysunek 5. Implementacja szyfru Gronsfelda w języku C#



Rysunek 6. Test implementacji

3.3 Szyfr transpozycyjny

Szyfr transpozycyjny to rodzaj szyfru, w którym zmienia się kolejność liter w wiadomości, ale same litery pozostają niezmienione. Zamiast zastępować litery innymi znakami (jak w szyfrze podstawieniowym), w szyfrze transpozycyjnym litery w tekście są permutowane (przemieszczane) w określony sposób zgodnie z ustaloną zasadą. Celem jest zamiana oryginalnej wiadomości w jej zaszyfrowaną formę, gdzie struktura tekstu, choć wciąż oparta na tych samych znakach, jest trudniejsza do odczytania bez znajomości klucza, który określa sposób permutacji.

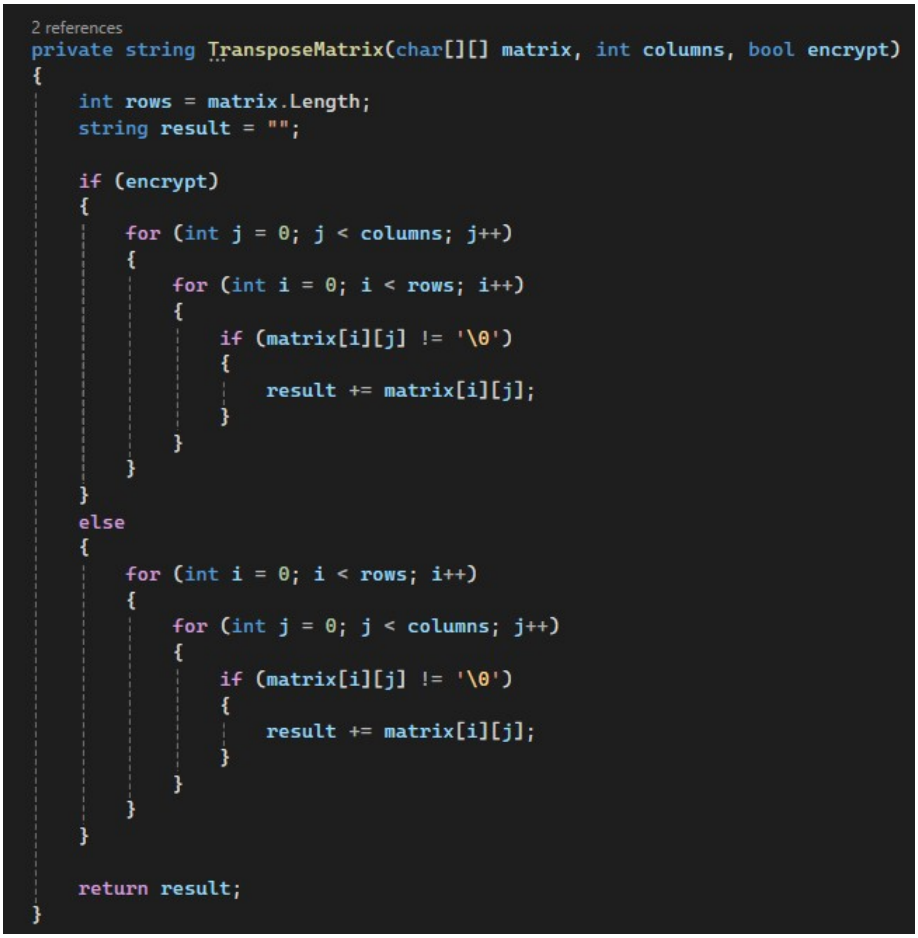
Szyfr transpozycyjny może przyjmować różne formy, takie jak szyfr kolumnowy czy szyfr permutacyjny. W szyfrze kolumnowym, na przykład, tekst jest zapisany w tabeli o określonej liczbie kolumn, a następnie odczytany w innej kolejności, co zmienia rozmieszczenie liter. Ważnym aspektem szyfru transpozycyjnego jest to, że każda litera oryginalnego tekstu pojawia się w zaszyfrowanej wiadomości, ale w zmienionej kolejności, a nie poprzez zmianę samej litery.

Funkcja przedstawiona poniżej na początku tworzy macierz z tekstu na podstawie liczby kolumn. Następnie oblicza liczbę wierszy macierzy na podstawie długości tekstu i liczby kolumn (za pomocą funkcji Math.Ceiling()). Potem inicjalizuje tablicę dwuwymiarową (char[][]), która reprezentuje macierz. Na końcu wypełnia macierz literami z tekstu, uzupełniając puste miejsca pustymi znakami ('\0'), jeśli tekst nie wypełnia dokładnie całej macierzy.

```
1 reference
private char[][] GenerateMatrix(string text, int columns)
{
    int rows = (int)Math.Ceiling((double)text.Length / columns);
    var matrix = new char[rows][];
    int charIndex = 0;

    for (int i = 0; i < rows; i++)
    {
        matrix[i] = new char[columns];
        for (int j = 0; j < columns; j++)
        {
            matrix[i][j] = charIndex < text.Length ? text[charIndex++] : '\0';
        }
    }
    return matrix;
}
```

Rysunek 7. Funkcja generująca macierz

Funkcja **TransposeMatrix** wykonuje transpozycję macierzy, czyli przekształca macierz w tekst, stosując różne metody w zależności od tego, czy szyfrujemy, czy odszyfrowujemy. Jeśli szyfrujemy (encrypt = true), funkcja iteruje przez kolumny, a następnie przez wiersze, tworząc ciąg zaszyfrowanego tekstu. Jeśli odszyfrowujemy (encrypt = false), funkcja iteruje przez wiersze, a następnie przez kolumny, tworząc odszyfrowany tekst.



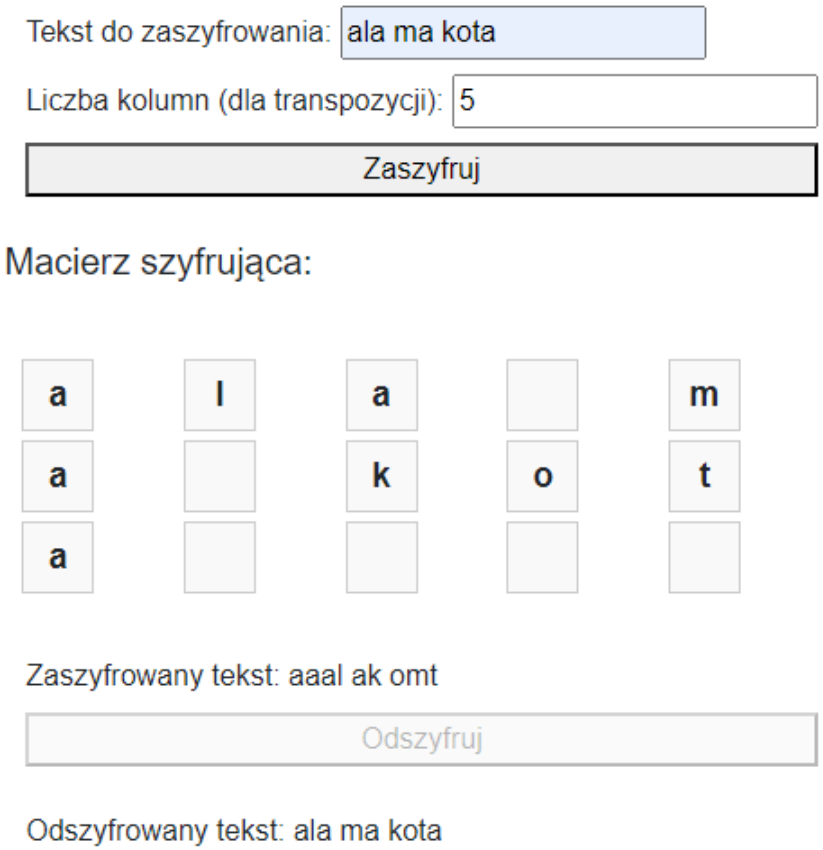
```
2 references
private string TransposeMatrix(char[][] matrix, int columns, bool encrypt)
{
    int rows = matrix.Length;
    string result = "";

    if (encrypt)
    {
        for (int j = 0; j < columns; j++)
        {
            for (int i = 0; i < rows; i++)
            {
                if (matrix[i][j] != '\0')
                {
                    result += matrix[i][j];
                }
            }
        }
    }
    else
    {
        for (int i = 0; i < rows; i++)
        {
            for (int j = 0; j < columns; j++)
            {
                if (matrix[i][j] != '\0')
                {
                    result += matrix[i][j];
                }
            }
        }
    }

    return result;
}
```

Rysunek 8. Funkcja przekształcająca macierz w tekst

Funkcja **GenerateDecryptionMatrix** tworzy macierz do deszyfrowania tekstu szyfrowanego szyfrem transpozycyjnym. Najpierw oblicza liczbę wierszy w macierzy oraz sposób rozdzielania znaków na kolumny (dzięki różnicy w wysokości kolumn). Następnie dzieli zaszyfrowany tekst na kolumny, umieszczając znaki w odpowiednich miejscach macierzy. Na końcu zależności od tego, czy kolumna jest pełna, czy nie, wypełnia odpowiednie wiersze macierzy.

```
1 reference
private char[][] GenerateDecryptionMatrix(string encryptedText, int columns)
{
    int rows = (int)Math.Ceiling((double)encryptedText.Length / columns);
    var matrix = new char[rows][];
    int charIndex = 0;

    int fullColumns = encryptedText.Length % columns;
    int fullColumnHeight = rows;
    int shortColumnHeight = rows - 1;

    for (int i = 0; i < rows; i++)
    {
        matrix[i] = new char[columns];
    }

    for (int col = 0; col < columns; col++)
    {
        int currentHeight = (col < fullColumns) ? fullColumnHeight : shortColumnHeight;
        for (int row = 0; row < currentHeight; row++)
        {
            if (charIndex < encryptedText.Length)
            {
                matrix[row][col] = encryptedText[charIndex++];
            }
        }
    }

    return matrix;
}
```

Rysunek 9. Funkcja tworząca macierz do deszyfrowania



Rysunek 10. Implementacja szyfru transpozycyjnego

3.4 Szyfrowanie blokowe - AES

3.5 Szyfrowanie blokowe - DES

3.6 Szyfrowanie strumieniowe - AES

3.7 Szyfrowanie strumieniowe - DES

3.8 Symulacja protokołu Diffiego-Hellmana

3.9 Szyfrowanie RSA

3.10 Certyfikat

3.11 Podpis cyfrowy

4. Podsumowanie i wnioski