

# Bezpieczne przechowywanie plików

*Stworzone przez*

Jakub72903 oraz Kacper72899

## Spis treści :

1. [Wymagania](#)
2. Struktura aplikacji
  - a. [app.py](#)
  - b. [Templates/index.html](#)
  - c. [Static/style.css](#)
3. [Testowanie](#)

## 1. Wymagania

Aplikacja wymaga użycia języka programowania [Python](#) w wersji [3.11](#), która oferuje najnowsze funkcje i usprawnienia języka, zapewniając wydajność i bezpieczeństwo kodu.

Do obsługi aplikacji internetowych wykorzystany zostanie [Flask](#) w wersji [2.3.2](#), który jest frameworkm do tworzenia aplikacji webowych. Flask zapewnia prostotę i elastyczność w projektowaniu aplikacji, oferując jednocześnie niezbędne narzędzia do zarządzania żądaniami, trasowaniem, szablonami i bezpieczeństwem.

## 2. Struktura aplikacji – app.py

Python

```
import os
import io
from flask import Flask, render_template, request, send_file, jsonify

app = Flask(__name__)
```

Importowane są niezbędne moduły: *os* dla operacji związanych z systemem operacyjnym, *io* dla operacji wejścia/wyjścia (np. obsługi strumieni), oraz kilka funkcji z framework'a Flask.

Inicjalizacja aplikacji Flask poprzez utworzenie instancji *Flask* i przypisanie jej do zmiennej *app*.

Python

```
def xor_operation(data, key):
    result_data = bytearray()
    for i in range(len(data)):
        data_byte = data[i]
        key_byte = key[i % len(key)]
        result_data.append(data_byte ^ key_byte)
    return bytes(result_data)
```

Definiuje funkcję *xor\_operation*, która przyjmuje dwa argumenty: *data* (dane do przetworzenia) i *key* (klucz do operacji XOR).

Dla każdego bajtu w *data* wykonuje operację XOR z odpowiadającym bajtem z *key* (z powtórzeniem klucza, jeśli jest krótszy niż dane).

Zwraca przetworzone dane jako obiekt *bytes*.

Python

```
@app.route('/', methods=['GET', 'POST'])

def index():

    if request.method == 'POST':

        operation = request.form.get('operation')
```

Dekorator `@app.route('/')` określa główną ścieżkę aplikacji obsługującą metody *GET* i *POST*. W funkcji *index*, sprawdzenie metody żądania (*POST*) inicjuje logikę przetwarzania, która zależy od typu operacji określonej przez użytkownika (*operation*).

Python

```
if operation == 'generate':

    key_string = request.form.get('key_string')

    if not key_string:

        return jsonify({"error": "Wymagany jest ciąg klucza"}), 400

    key_path = request.form.get('key_path', 'klucz.txt')

    if not key_path.endswith('.txt'):

        key_path += '.txt'
```

```
with open(key_path, "w") as key_file:  
    key_file.write(key_string)  
  
return jsonify({"message": "Klucz wygenerowany", "key_path":  
    key_path})
```

Logika dla operacji *generate* obejmuje weryfikację czy istnieje *key\_string*, określenie ścieżki zapisu klucza (*key\_path*) z domyślną wartością i ewentualnym dodaniem rozszerzenia *.txt*, zapisanie klucza do pliku i zwrócenie informacji o sukcesie operacji.

Python

```
elif operation == 'load':  
    return load_key_file()
```

Dla operacji *load*, wywoływana jest funkcja *load\_key\_file* do obsługi wczytywania i zwracania zawartości pliku klucza.

Python

```
elif operation in ['encrypt', 'decrypt']:  
    key_file = request.files.get('key_file')  
    if not key_file:  
        return jsonify({"error": "Wymagany jest klucz"}), 400  
  
    key = key_file.read()  
    file = request.files.get('file')  
    if not file:  
        return jsonify({"error": "Wymagany jest plik"}), 400  
  
    file_data = file.read()  
    processed_data = xor_operation(file_data, key)
```

```
filename = 'Zaszyfrowany.txt' if operation == 'encrypt' else  
'Odszyfrowany.txt'  
  
return send_file(io.BytesIO(processed_data),  
download_name=filename, as_attachment=True)
```

Logika dla operacji *encrypt* i *decrypt* obejmuje weryfikację dostępności przesłanych plików (*key\_file* i plik do przetworzenia), wykonanie operacji XOR na danych pliku przy użyciu klucza, a następnie zwrócenie przetworzonych danych jako pliku do pobrania, z nazwą zależną od typu operacji.

Python

```
def load_key_file():  
    try:  
        if 'key_file' not in request.files:  
            return jsonify({"error": "Wymagany jest klucz"}), 400  
  
        key_file = request.files['key_file']  
        if key_file.filename == '':  
            return jsonify({"error": "Wymagany jest plik"}), 400  
  
        key = key_file.read()  
        return jsonify({"message": "Klucz został pomyślnie wczytany",  
"key": key.hex()})
```

Funkcja *load\_key\_file* rozpoczyna się od bloku *try*, który ma na celu obsługę potencjalnych wyjątków, które mogą wystąpić podczas wczytywania pliku klucza.

Funkcja najpierw sprawdza, czy w przesłanych plikach (*request.files*) znajduje się plik o nazwie *key\_file*. Jeśli nie, zwracany jest błąd JSON z komunikatem, że wymagany jest klucz, wraz z kodem błędu *HTTP 400*.

Następnie weryfikowana jest nazwa pliku klucza. Jeśli nazwa pliku jest pusta (co oznacza, że plik nie został przesłany), funkcja również zwraca błąd JSON z komunikatem o wymaganym pliku, z kodem błędu 400.

Python

```
except Exception as e:  
    return jsonify({"error": str(e)}), 500
```

W przypadku wystąpienia wyjątku podczas wykonywania operacji wczytywania pliku, blok `except` przechwytuje wyjątek i zwraca ogólny komunikat o błędzie wraz z kodem błędu HTTP 500. To informuje użytkownika o wewnętrznym błędzie serwera, nie ujawniając szczegółów wyjątku, co może być korzystne z punktu widzenia bezpieczeństwa aplikacji.

Python

```
if __name__ == '__main__':  
    app.run(debug=True)
```

Uruchamia serwer aplikacji w trybie `debug`

## 2. Struktura aplikacji - index.html

HTML

```
<!DOCTYPE html>

<head>
    <meta charset="UTF-8">
    <title>Bezpieczne trzymanie plików</title>
    <link rel="stylesheet" href="{{ url_for('static',
        filename='style.css') }}">
</head>
```

Deklaracja typu dokumentu HTML5 (`<!DOCTYPE html>`), sekcja `<head>` zawierająca metadane takie jak kodowanie znaków (*UTF-8*), tytuł strony (*Bezpieczne trzymanie plików*) oraz odnośnik do zewnętrznego arkusza stylów CSS (*style.css*), którego ścieżka jest dynamicznie generowana przez Flask.

HTML

```
<body>
    <div class="container">
        <h1>Bezpieczne trzymanie plików</h1>
```

Początek sekcji `<body>` zawierającej główny kontener strony (`<div class="container">`). Nagłówek `<h1>` określa tytuł strony, który jest widoczny dla użytkownika jako "Bezpieczne trzymanie plików".

## HTML

```
<h2>Generowanie</h2>

<form action="/" method="post">

    <input type="hidden" name="operation" value="generate">

    <h3>Wpisz klucz</h3>

    <input type="text" name="key_string" placeholder="Wpisz klucz"
           required>

    <h3>Podaj ścieżkę zapisu klucza</h3>

    <input type="text" name="key_path" placeholder="C:\...">

    <button type="submit">Generuj klucz</button>

</form>
```

Sekcja z nagłówkiem `<h2>Generowanie</h2>` zawiera formularz do generowania klucza. Formularz ten przesyła dane metodą *POST* na główną ścieżkę (`action="/"`). Ukryte pole ***operation*** służy do określenia typu operacji. Dwa pola tekstowe pozwalają użytkownikowi na wprowadzenie klucza (***key\_string***) i opcjonalnie określenie ścieżki zapisu klucza (***key\_path***). Przycisk ***submit*** inicjuje akcję generowania klucza.

## HTML

```
<h2>Szyfrowanie</h2>

<form action="/" method="post" enctype="multipart/form-data">

    <input type="hidden" name="operation" value="encrypt">

    <h3>Klucz</h3>

    <input type="file" name="key_file" required>

    <h3>Plik do zaszyfrowania</h3>

    <input type="file" name="file" required>

    <button type="submit">Szyfruj plik</button>

</form>
```

Sekcja z nagłówkiem `<h2>Szyfrowanie</h2>` zawiera formularz umożliwiający szyfrowanie plików. Podobnie jak wcześniej, używa metody *POST* i zawiera ukryte pole ***operation*** z wartością "encrypt". Dwa pola typu ***file*** umożliwiają wybór pliku klucza

(*key\_file*) i pliku do zaszyfrowania (*file*). Oba pola są oznaczone jako wymagane (*required*). Przycisk *submit* uruchamia proces szyfrowania.

HTML

```
<h2>Deszyfrowanie</h2>

<form action="/" method="post" enctype="multipart/form-data">
    <input type="hidden" name="operation" value="decrypt">
    <h3>Klucz</h3>
    <input type="file" name="key_file" required>
    <h3>Plik do deszyfrowania</h3>
    <input type="file" name="file" required>
    <button type="submit">Deszyfruj plik</button>
</form>
</div>
</body>
</html>
```

Ostatnia sekcja z nagłówkiem **<h2>Deszyfrowanie</h2>** zawiera formularz do deszyfrowania plików. Struktura jest analogiczna do formularza szyfrowania, z tą różnicą, że ukryte pole *operation* ma wartość "decrypt". Pola typu *file* pozwalają na wybór pliku klucza (*key\_file*) i pliku do deszyfrowania (*file*). Przycisk *submit* inicjuje proces deszyfrowania.

## 2. Struktura aplikacji – style.css

css

```
body {  
    font-family: Arial, sans-serif;  
    display: flex;  
    justify-content: center;  
    align-items: flex-start;  
    height: 100vh;  
    margin: 0;  
    background-color: #f0e3d8;  
    padding-top: 50px;  
}
```

Styl dla całego ciała dokumentu ustawia czcionkę na Arial, wyrównuje treść na środku (poziomo) i na początku kontenera (pionowo). Wysokość jest ustawiona na 100% widoku przeglądarki (100vh), z brakiem marginesu, jasnobrązowym kolorem tła (#f0e3d8) i odstępem od górnej krawędzi o 50px.

css

```
.container {  
    background-color: white;  
    padding: 20px;  
    border-radius: 8px;  
    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);  
    max-width: 90%;  
    margin: 0 auto;  
    display: flex;  
    flex-direction: column;  
    align-items: center;  
    gap: 20px;  
}
```

Styl dla elementu `.container` definiuje biały kolor tła, padding, zaokrąglenie rogów, subtelne cieniowanie, maksymalną szerokość na 90% kontenera, automatyczne wyśrodkowanie, układ kolumnowy Flexbox z elementami wyrównanymi do środka i odstępem 20px między nimi.

CSS

```
.section-frame {  
    width: 100%;  
    max-width: 600px;  
    margin: 0 auto;  
    padding: 20px;  
    border: 1px solid #ddd;  
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);  
    border-radius: 10px;  
}
```

Styl `.section-frame` określa ramki sekcji z pełną szerokością, maksymalną szerokością 600px, wyśrodkowaniem, paddingiem, cienką, stałą ramką, jaśniejszym cieniowaniem niż `.container`, i większym zaokrągleniem rogów.

CSS

```
form {  
    display: flex;  
    flex-direction: column;  
    width: 300px;  
}
```

Styl dla elementów `form` ustala układ Flexbox w kierunku kolumny, z szerokością formularza ustawioną na 300px.

css

```
form h2 {  
    margin-top: 0;  
}
```

Usuwa górny margines z nagłówków `<h2>` wewnątrz formularzy, aby zapewnić spójność i estetykę.

css

```
label {  
    margin-top: 10px;  
}
```

Dodaje górny margines do wszystkich etykiet (`label`) w formularzach, poprawiając czytelność przez zapewnienie przestrzeni między polami formularza.

css

```
input[type="text"], input[type="file"], select {  
    padding: 8px;  
    margin-top: 5px;  
    border: 1px solid #ddd;  
    border-radius: 4px;  
}
```

Stylizuje pola tekstowe, plikowe i rozwijane, dodając padding, mały górny margines, cienką ramkę i lekko zaokrąglone rogi.

css

```
button {  
    margin-top: 20px;  
    padding: 10px;  
    border: none;  
    border-radius: 4px;  
    background-color: #007bff;  
    color: white;  
    cursor: pointer;  
}
```

Stylizuje przyciski, nadając im górny margines, padding, brak ramki, zaokrąglenie rogów, niebieski kolor tła, biały tekst i kursor wskazujący na możliwość kliknięcia.

css

```
button:hover {  
    background-color: #0056b3;  
}
```

Reguła **button:hover** nadaje efekt najechania myszką na przyciski, zmieniając ich kolor tła na ciemniejszy odcień niebieskiego,

### 3. Testowanie

#### Generowanie

**Wpisz klucz**

**Podaj ścieżkę zapisu klucza**

**Generuj klucz**

Użytkownik wchodzi na stronę główną i widzi formularz "**Generowanie**", gdzie wpisuje klucz i określa ścieżkę, precyzując gdzie klucz zostanie zapisany na dysku. Po wypełnieniu formularza i kliknięciu "Generuj klucz", aplikacja Flask przetwarza żądanie **POST**, tworzy plik klucza i zwraca komunikat JSON z potwierdzeniem oraz ścieżką do wygenerowanego klucza.

```
{  
    "key_path": "H:\\projekt\\klucz.txt",  
    "message": "Klucz wygenerowany"  
}
```

# Szyfrowanie

## Klucz

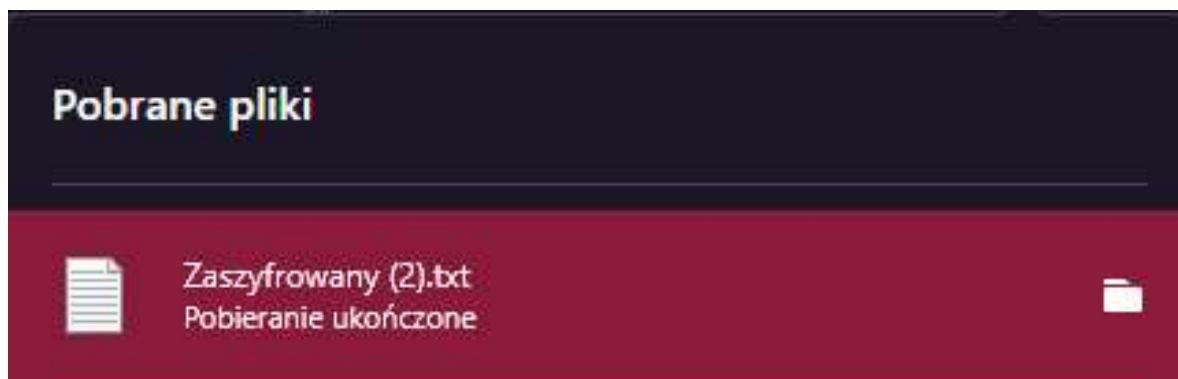
Wybierz plik klucz.txt

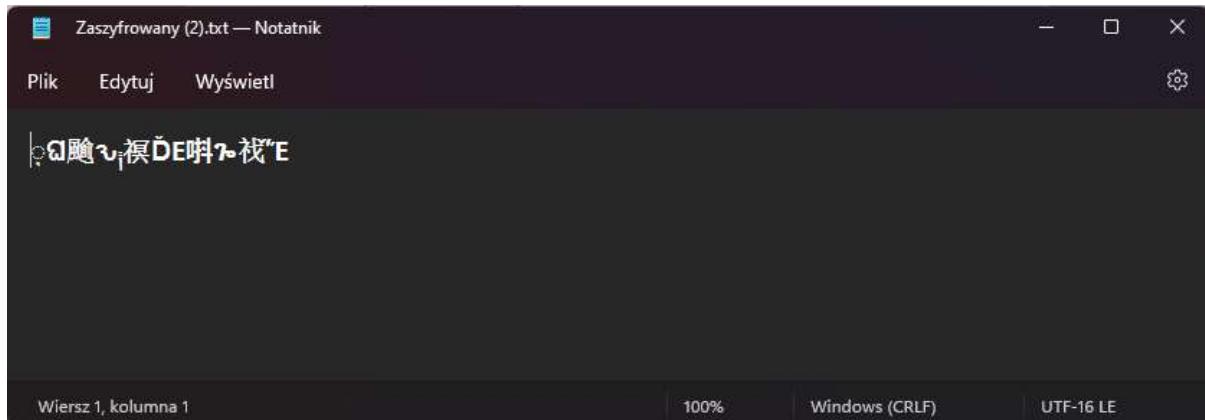
## Plik do zaszyfrowania

Wybierz plik plik.txt

Szyfruj plik

W kolejnym kroku, użytkownik ma możliwość zaszyfrowania plików. Wybiera on plik klucza oraz plik przeznaczony do zaszyfrowania za pomocą formularza „Szyfrowanie”. Po wysłaniu formularza, aplikacja wykonuje operację szyfrowania XOR na wybranym pliku danych, korzystając z wybranego klucza. Zaszyfrowany plik jest następnie automatycznie pobierany na lokalny dysk użytkownika.





Na powyższym zrzucie, widać wnętrze pliku zaszyfrowanego.

# Deszyfrowanie

Klucz

Wybierz plik klucz.txt

## Plik do deszyfrowania

Wybierz plik Zaszyfrowany (2).txt

## Deszyfruj plik

W ostatniej części procesu, użytkownik może odszyfrować zaszyfrowany plik. Użytkownik wybiera ***ten sam*** plik klucza i zaszyfrowany plik w formularzu "***Deszyfrowanie***". Po przesłaniu formularza aplikacja przeprowadza operację deszyfrowania i zwraca odszyfrowany plik, który użytkownik może pobrać.

Odszyfrowany tekst, "Lorem ipsum dolor sit amet", jest widoczny poniżej, co potwierdza sukces operacji deszyfrowania.

