

Dokumentacja Projektu CryptoLab Mobile

Agnieszka Ryś

20 października 2025

Spis treści

1 Cel projektu

Celem aplikacji **CryptoLab Mobile** jest edukacja w zakresie kryptografii. Aplikacja mobilna pozwala szyfrować i deszyfrować teksty oraz pliki `.txt`, sprawdzać poprawność kluczy oraz eksportować wyniki. Wszystkie algorytmy są implementowane ręcznie, bez użycia gotowych bibliotek kryptograficznych.

2 Podstawy kryptografii klasycznej

Wprowadzenie

Kryptografia to dziedzina zajmująca się ochroną informacji poprzez jej przekształcanie w formę nieczytelną dla osób nieuprawnionych. Jej historia sięga starożytności, gdzie stosowano proste metody szyfrowania, znane obecnie jako **kryptografia klasyczna**. Celem było zapewnienie poufności korespondencji wojskowej, dyplomatycznej czy handlowej.

2.1 Kryptografia symetryczna

W kryptografii symetrycznej ten sam klucz służy zarówno do szyfrowania, jak i deszyfrowania wiadomości. Najważniejsze cechy:

- wysoka szybkość działania,
- konieczność bezpiecznej wymiany klucza,
- podatność na ataki brute-force przy krótkich kluczach.

2.2 Przykłady szyfrów klasycznych

- **Szyfr Cezara** – przesunięcie liter alfabetu o stałą liczbę pozycji,
- **Szyfr Vigenère’a** – wieloalfabetyczny szyfr wykorzystujący słowo-klucz,
- **Szyfr z kluczem bieżącym** – rozwinięcie Vigenère’a z długim kluczem tekstowym,
- **Szyfr podstawieniowy (monoalfabetyczny)** – każdej literze alfabetu przypisana jest inna litera,
- **Szyfr Playfair** – operujący na parach liter,
- **Szyfr transpozycyjny** – zmienia kolejność znaków w wiadomości.

2.3 Znaczenie w edukacji

Choć współcześnie klasyczne szyfry nie zapewniają realnego bezpieczeństwa, stanowią doskonale narzędzie dydaktyczne. Pozwalają zrozumieć podstawowe pojęcia kryptografii, takie jak:

- **klucz** – parametr definiujący szyfrowanie,
- **przestrzeń kluczy** – zbiór możliwych wartości klucza,

- **analiza częstości** – klasyczna metoda łamania szyfrów,
- **brute-force** – przeszukiwanie wszystkich możliwych kluczy.

3 Miejsce szyfru Cezara

Szyfr Cezara należy do najprostszych szyfrów podstawieniowych. Choć jego bezpieczeństwo jest znikome, odgrywa on kluczową rolę w nauczaniu, ponieważ wprowadza intuicyjnie pojęcia klucza, szyfrowania i deszyfrowania. CryptoLab wykorzystuje go jako **pierwszy krok** w implementacji i analizie algorytmów kryptograficznych.

4 Technologie wykorzystane w projekcie

React Native + Expo Główna platforma wykorzystana do tworzenia aplikacji mobilnych. React Native umożliwia budowanie natywnych aplikacji na systemy Android i iOS, wykorzystując składnię zbliżoną do Reacta. Expo zostało użyte jako narzędzie wspierające proces developmentu – upraszcza konfigurację środowiska, przyspiesza testowanie na urządzeniach mobilnych i zapewnia dostęp do bogatego ekosystemu bibliotek.

TypeScript Nadzbiór JavaScriptu wprowadzający system typów. Zastosowanie TypeScriptu pozwoliło na:

- wcześniejsze wykrywanie błędów podczas kompilacji,
- lepszą kontrolę nad strukturą danych i interfejsami,
- zwiększoną czytelność oraz przewidywalność kodu,

Expo Document Picker, File System, Vector Icons Dodatkowe biblioteki środowiska Expo:

- **expo-document-picker** – umożliwia wybór plików z pamięci urządzenia,
- **expo-file-system** – zapewnia dostęp do systemu plików (zapisywanie, odczyt, usuwanie plików),
- **expo-vector-icons** – biblioteka ikon pozwalająca wzbogacić interfejs użytkownika.

Git System kontroli wersji użyty do zarządzania historią kodu. Pozwolił na prowadzenie szczegółowego changelogu, śledzenie postępów w projekcie oraz łatwe zarządzanie zmianami w kodzie źródłowym.

LaTeX System składu tekstu wykorzystany do przygotowania dokumentacji. Umożliwia on:

- zachowanie spójności formatowania,
- wygodne dodawanie fragmentów kodu źródłowego i zrzutów ekranu,
- automatyczne generowanie spisów treści i numeracji.

5 Architektura systemu

5.1 Wzorzec projektowy

Aplikacja wykorzystuje **Strategy Pattern** dla algorytmów kryptograficznych. Każdy algorytm dziedziczy z klasy abstrakcyjnej `CryptographicAlgorithm` i implementuje metody:

- `encrypt(plaintext, key)` – szyfruje tekst,
- `decrypt(ciphertext, key)` – deszyfruje tekst,
- `validateKey(key)` – sprawdza poprawność klucza,
- `getKeyRequirements()` – zwraca opis wymagań dla klucza.

Wszystkie algorytmy zarejestrowane są w `AlgorithmRegistry` (Singleton Pattern), co umożliwia łatwe dodawanie nowych szyfrów bez modyfikacji głównej aplikacji.

5.2 Komponenty główne

- `App.tsx` – główny komponent aplikacji, obsługuje interfejs użytkownika,
- `AlgorithmSidebar.tsx` – boczny panel z listą dostępnych algorytmów,
- `AlgorithmRegistry.ts` – rejestr i zarządzanie algorytmami,
- `CryptographicAlgorithm.ts` – klasa bazowa dla wszystkich algorytmów,
- `fileUtils.ts` – funkcje do obsługi operacji na plikach.

6 Struktura projektu

```
crypto-lab-mobile/
  App.tsx                (główny komponent)
  package.json           (zależności projektu)
  tsconfig.json          (konfiguracja TypeScriptu)
  app.json               (konfiguracja Expo)
  src/
    algorithms/
      CryptographicAlgorithm.ts (klasa bazowa)
      CaesarCipher.ts          (szyfr Cezara)
      VigenereCipher.ts        (szyfr Vigenere'a)
      RunningKeyCipher.ts      (szyfr z kluczem
    bieżącym)
      AlgorithmRegistry.ts     (rejestr algorytmów)
    components/
      AlgorithmSidebar.tsx     (panel z algorytmami)
    utils/
      fileUtils.ts             (obsługa plików)
  assets/                   (zasoby graficzne)
```

7 Implementacja szyfru Cezara

7.1 Podstawy

Szyfr Cezara to prosty szyfr monoalfabetyczny, w którym litery przesuwane są o wartość klucza k . Przestrzeń kluczy obejmuje wartości 1–25. Metoda jest podatna na ataki brute-force i analizę częstotliwości.

7.2 Model matematyczny

- Szyfrowanie: $E_k(x) = (x + k) \bmod 26$,
- Deszyfrowanie: $D_k(x) = (x - k) \bmod 26$.

7.3 Cechy implementacji

- Obsługuje zarówno wielkie jak i małe litery,
- Znaki niebędące literami pozostają bez zmian,
- Klucz musi być liczbą całkowitą z zakresu 1–25,
- Walidacja klucza zwraca szczegółową informację o błędach.

8 Implementacja szyfru Vigenère’a

8.1 Historia i znaczenie

Szyfr Vigenère’a został opracowany w XVI wieku przez Blaise de Vigenère’a. Przez długi czas uważany był za niezniszczalny (*le chiffre indéchiffrable*) aż do jego przełamania przez Charles’a Babbage’a w XIX wieku.

8.2 Podstawy

Szyfr Vigenère’a to szyfr **polialfabetyczny**, który wykorzystuje słowo-klucz do generowania serii przesunięć. W przeciwieństwie do szyfru Cezara, każda litera tekstu może być szyfrowana z innym przesunięciem.

8.3 Model matematyczny

- Szyfrowanie: $E_k(x_i) = (x_i + k_{i \bmod |k|}) \bmod 26$,
- Deszyfrowanie: $D_k(y_i) = (y_i - k_{i \bmod |k|}) \bmod 26$,
- gdzie k to słowo-klucz, a $|k|$ to jego długość.

8.4 Przykład działania

Tekst jawny	A	T	T	A	C	K
Klucz	L	E	M	O	N	L
Przesunięcia	+11	+4	+12	+14	+13	+11
Tekst zaszyfrowany	L	X	F	O	P	V

8.5 Cechy implementacji

- Klucz może zawierać tylko litery (A-Z, a-z),
- Klucz nie może być pusty,
- Znaki niebędące literami w tekście źródłowym są przepisywane bez zmian,
- Klucz automatycznie się powtarza dla długich tekstów,
- Obsługuje zarówno wielkie jak i małe litery w tekście.

9 Implementacja szyfru z kluczem bieżącym

9.1 Historia i zastosowanie

Szyfr z kluczem bieżącym (Running Key Cipher) to rozwinięcie szyfru Vigenère’a. Zamiast krótko słowa, wykorzystuje on klucz o długości co najmniej równej długości tekstu. Gdy klucz jest naprawdę losowy i będzie użyty tylko raz, szyfr ten jest teoretycznie nie do złamania (jest to wariant szyfru jednorazowego – *One-Time Pad*).

9.2 Podstawy

Algorytm jest w zasadzie identyczny z szyfrem Vigenère’a, ale z istotną różnicą: klucz powinien być znacznie dłuższy niż tekst. W praktyce zastosowania edukacyjnego aplikacja automatycznie generuje klucz z tekstu Lorem Ipsum.

9.3 Model matematyczny

- Szyfrowanie: $E_k(x_i) = (x_i + k_i) \bmod 26$,
- Deszyfrowanie: $D_k(y_i) = (y_i - k_i) \bmod 26$,
- gdzie $|k| \geq |x|$ (klucz jest co najmniej tak długi jak tekst).

9.4 Cechy implementacji

- Klucz może zawierać litery i spacje,
- Klucz musi zawierać co najmniej 5 liter,
- Aplikacja automatycznie generuje losowy klucz na bazie Lorem Ipsum,
- Zaszyfrowany tekst zawiera klucz w formacie: `<klucz>::<tekst_zaszyfrowany>`,
- Deszyfrowanie wymaga podania tekstu w poprawnym formacie.

9.5 Bezpieczeństwo

- Gdy klucz jest losowy i używany tylko raz, szyfr jest teoretycznie bezpieczny,
- Słaba strona: jeśli klucz jest krótszy niż tekst, powtarza się i traci bezpieczeństwo,
- W aplikacji edukacyjnej klucz jest generowany automatycznie i przechowywany w wynikach.

10 Wybrane fragmenty kodu

10.1 Klasa bazowa algorytmu

Listing 1: Klasa abstrakcyjna CryptographicAlgorithm

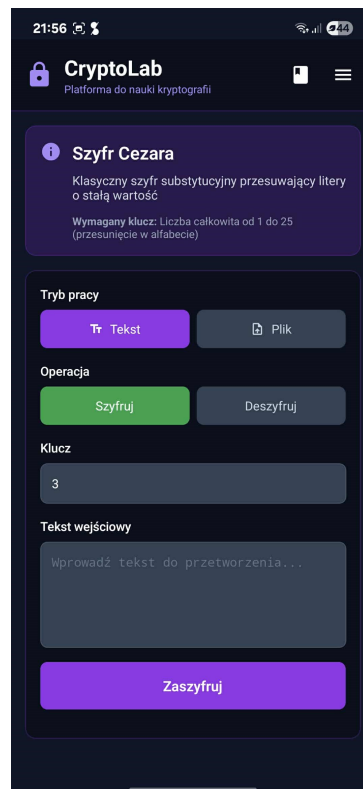
```
export default class CryptographicAlgorithm {
  name: string;
  description: string;
  category: string;

  encrypt(plaintext: string, key: string): string {
    throw new Error('Metoda encrypt() musi by zaimplementowana');
  }

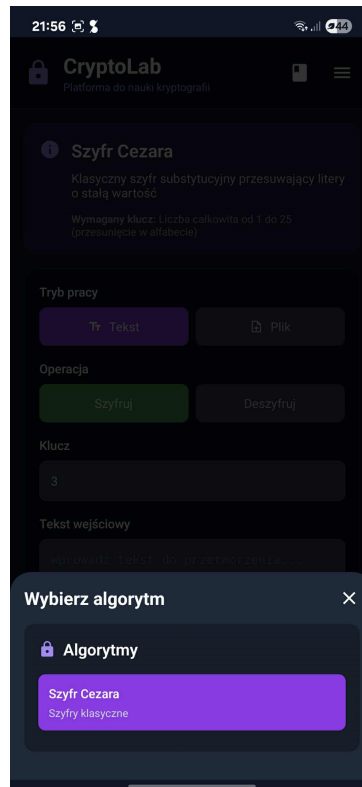
  decrypt(ciphertext: string, key: string): string {
    throw new Error('Metoda decrypt() musi by zaimplementowana');
  }

  validateKey(key: string): { valid: boolean; error?: string } {
    throw new Error('Metoda validateKey() musi by zaimplementowana');
  }

  getKeyRequirements(): string {
    throw new Error('Metoda getKeyRequirements() musi by
      zaimplementowana');
  }
}
```



Rysunek 1: Ekran główny aplikacji CryptoLab



Rysunek 2: Lista z możliwością wyboru algorytmu

10.2 Implementacja szyfru Cezara

Listing 2: Szczegóły implementacji CaesarCipher

```
export default class CaesarCipher extends CryptographicAlgorithm {
  constructor() {
    super(
      'Szyfr Cezara',
      'Prosty szyfr substytucyjny z przesunięciem',
      'Szyfry klasyczne'
    );
  }

  validateKey(key: string): { valid: boolean; error?: string } {
    const numKey = parseInt(key, 10);
    if (isNaN(numKey) || numKey < 1 || numKey > 25) {
      return {
        valid: false,
        error: 'Klucz musi być liczbą od 1 do 25'
      };
    }
    return { valid: true };
  }

  encrypt(plaintext: string, key: string): string {
    return this._process(plaintext, parseInt(key, 10));
  }

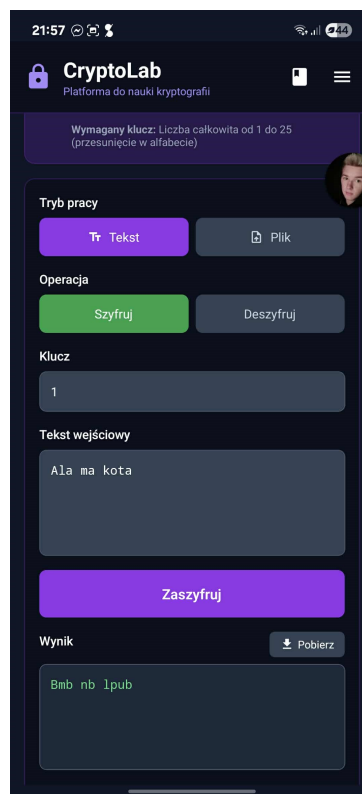
  decrypt(ciphertext: string, key: string): string {
    const shift = 26 - (parseInt(key, 10) % 26);
  }
}
```

```

    return this._process(ciphertext, shift);
}

private _process(text: string, shift: number): string {
    return text.split('').map(char => {
        if (/[A-Za-z]/.test(char)) {
            const base = char === char.toUpperCase() ? 65 : 97;
            return String.fromCharCode(
                (char.charCodeAt(0) - base + shift) % 26 + base
            );
        }
        return char;
    }).join('');
}
}

```



Rysunek 3: Test szyfru Cezara

10.3 Implementacja szyfru Vigenère'a

Listing 3: Fragmenty klasy VigenereCipher

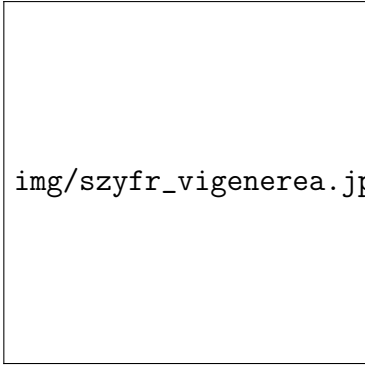
```
export default class VigenereCipher extends CryptographicAlgorithm {
  validateKey(key: string): { valid: boolean; error?: string } {
    if (!key || key.trim().length === 0) {
      return { valid: false, error: 'Klucz nie może być pusty' };
    }
    const hasOnlyLetters = /^[a-zA-Z]+$/.test(key);
    if (!hasOnlyLetters) {
      return { valid: false, error: 'Klucz może zawierać tylko litery' };
    }
    return { valid: true };
  }

  private _process(text: string, key: string, encrypt: boolean): string {
    {
      let result = '';
      let keyIndex = 0;
      const normalizedKey = key.toUpperCase();

      for (let i = 0; i < text.length; i++) {
        const char = text[i];
        if (/[A-Za-z]/.test(char)) {
          const base = char === char.toUpperCase() ? 65 : 97;
          const textCode = char.charCodeAt(0) - base;
          const keyCode = normalizedKey.charCodeAt(keyIndex %
            normalizedKey.length) - 65;

          const resultCode = encrypt
            ? (textCode + keyCode) % 26
            : (textCode - keyCode + 26) % 26;

          result += String.fromCharCode(resultCode + base);
          keyIndex++;
        } else {
          result += char;
        }
      }
      return result;
    }
  }
}
```



img/szyfr_vigenerea.jpg

Rysunek 4: Ekran szyfru Vigenere'a

10.4 Implementacja szyfru z kluczem bieżącym

Listing 4: Fragmenty klasy RunningKeyCipher

```
export default class RunningKeyCipher extends CryptographicAlgorithm {
  constructor() {
    super(
      'Szyfr z kluczem bieżącym',
      'Szyfr podobny do Vigenere\'a, ale używaj ci klucza o długości tekstu',
      'Szyfry klasyczne'
    );
  }

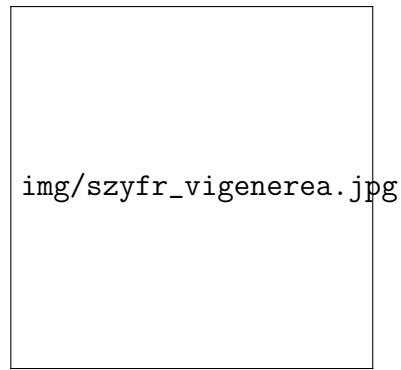
  validateKey(key: string): { valid: boolean; error?: string } {
    if (!key || key.trim().length === 0) {
      return { valid: false, error: 'Klucz nie może być pusty' };
    }

    // Sprawdź czy klucz zawiera tylko litery
    const hasOnlyLetters = /^[a-zA-Z\s]+$/.test(key);
    if (!hasOnlyLetters) {
      return { valid: false, error: 'Klucz może zawierać tylko litery i spacje (A-Z, a-z)' };
    }

    // Policz tylko litery w kluczu
    const keyLettersCount = key.replace(/[^a-zA-Z]/g, '').length;
    if (keyLettersCount < 5) {
      return {
        valid: false,
        error: 'Klucz musi zawierać co najmniej 5 liter (może zawierać spacje)'
      };
    }

    return { valid: true };
  }

  getKeyRequirements(): string {
    return 'Tekst (np. fragment księgi) - użyto generatora lorem ipsum do stworzenia klucza';
  }
}
```



Rysunek 5: Ekran szyfru z kluczem bieżącym

11 Podsumowanie

11.1 Szyfr Cezara

Szyfr Cezara należy do najstarszych i najprostszych technik szyfrowania. Jego główna idea polega na przesuwaniu liter alfabetu o ustaloną liczbę pozycji. Mimo że w praktyce jest to jedynie przykład historyczny, implementacja szyfru pozwala lepiej zrozumieć podstawowe mechanizmy kryptografii, takie jak klucz, szyfrowanie i deszyfrowanie.

Zalety:

- bardzo prosta implementacja,
- szybkie działanie,
- dobre ćwiczenie dydaktyczne.

Wady:

- niska odporność na ataki kryptograficzne,
- atak brute-force łatwo przełamuje szyfr w sekundach,
- podatny na analizę częstotliwości.

11.2 Szyfr Vigenère’a

Szyfr Vigenère’a to znacznie bardziej zaawansowany szyfr polialfabetyczny. Przez wieki uważany był za niezniszczalny, ale ostatecznie został przełamany dzięki analizie częstotliwości długości okresu.

Zalety:

- znacznie bardziej bezpieczny niż szyfr Cezara,
- odporne na prostą analizę częstotliwości,
- wykorzystuje koncepcję słowa-klucza, co jest intuicyjne.

Wady:

- niska odporność na ataki kryptograficzne (możliwy atak siłowy poprzez sprawdzenie wszystkich przesunięć),
- brak zastosowania we współczesnych systemach bezpieczeństwa,
- szyfr działa jedynie na ograniczonym zbiorze znaków (najczęściej alfabet łaciński).

11.3 Szyfr z kluczem bieżącym

Szyfr z kluczem bieżącym to krok w kierunku szyfrowania jednorazowego.

Zalety:

- gdy klucz jest losowy i używany raz – teoretycznie nie do złamania,
- koncepcja zbliża się do rzeczywistego bezpieczeństwa informacyjnego,
- edukacyjnie pokazuje znaczenie losowości klucza.

Wady:

- wymaga przechowywania bardzo długich kluczy,
- wymaga absolutnej losowości i jednorazowego użycia,
- niepraktyczne w większości rzeczywistych zastosowań.

12 Changelog

- **14.10.2025** Implementacja szyfru Cezara (szyfrowanie, deszyfrowanie, walidacja klucza) oraz podstawowe GUI.
- **20.10.2025** Dodanie szyfru Vigenère’a i szyfru z kluczem bieżącym.
Ulepszenie interfejsu użytkownika.
Implementacja AlgorithmRegistry z wzorcem Singleton.
Ulepszenie walidacji kluczy z szczegółowymi komunikatami o błędach.