

AKADEMIA NAUK STOSOWANYCH W NOWYM SĄCZU

Wydział Nauk Inżynieryjnych
Katedra Informatyki

DOKUMENTACJA

Kryptografia i teoria kodów

Algorytmy Kryptograficzne

Autor:

Łukasz Nowak

Prowadzący:

mgr inż. Drozd Daniel

Nowy Sącz 2025

Spis treści

| | |
|--|-----------|
| 1. Cel projektu | 3 |
| 2. Algorytmy Kryptograficzne | 4 |
| 2.1. Szyfr Cezara | 4 |
| 2.1.1. Zasada działania | 4 |
| 2.1.2. Przykład działania | 4 |
| 2.2. Podsumowanie | 4 |
| 2.3. Running Key Cipher | 5 |
| 2.3.1. Zasada działania | 5 |
| 2.3.2. Podsumowanie | 5 |
| 2.4. Szyfr Beauforta | 6 |
| 2.4.1. Zasada działania | 6 |
| 2.4.2. Podsumowanie | 6 |
| 3. Technologie | 7 |
| 3.1. Python | 7 |
| 3.2. Tkinter | 7 |
| 3.3. System kontroli wersji – GitHub | 8 |
| 4. Architektura oraz wygląd aplikacji | 9 |
| 4.1. Architektura techniczna | 9 |
| 4.1.1. Przykładowe fragmenty kodu | 10 |
| 4.2. Wygląd aplikacji | 10 |
| 4.3. Aktualizacje wyglądu | 11 |
| 5. Changelog | 13 |
| Literatura | 14 |
| Spis rysunków | 14 |
| Spis tabel | 15 |
| Spis listingów | 16 |

1. Cel projektu

Celem projektu jest stworzenie aplikacji okienkowej, która będzie uniwersalną bazą do rozwijania i testowania różnych algorytmów kryptograficznych. Aplikacja ma umożliwiać szyfrowanie i deszyfrowanie zarówno tekstu, jak i plików z wykorzystaniem kolejnych, implementowanych modułowo algorytmów. Każdy z dodawanych algorytmów będzie zaimplementowany jako osobna klasa lub moduł, zgodnie z zasadami dobrej architektury oprogramowania i wzorcami projektowymi umożliwiającymi łatwą rozbudowę projektu w przyszłości.

Aplikacja służyć będzie jako narzędzie dydaktyczne do praktycznego poznawania technik szyfrowania oraz omówienia zagadnień związanych z kryptografią klasyczną i współczesną. Dokumentacja projektu obejmuje opisy zastosowanych rozwiązań, szczegółowe wyjaśnienia wybranych metod kryptografii oraz dokładny changelog każdej wprowadzonej zmiany.

2. Algorytmy Kryptograficzne

2.1. Szyfr Cezara

Szyfr Cezara to klasyczny algorytm kryptograficzny, zaliczany do szyfrów podstawieniowych, który polega na przesuwaniu każdej litery w tekście jawnego o stałą liczbę pozycji. Klucz szyfrowania (przesunięcie) jest liczbą określającą, o ile miejsc każda litera zostanie przesunięta w alfabecie.

2.1.1. Zasada działania

Niech:

- x – numer pozycji litery w alfabecie (np. $A = 0, B = 1, \dots, Z = 25$),
- k – klucz przesunięcia,
- y – numer pozycji litery po zaszyfrowaniu.

Szyfrowanie zapisujemy wzorem: $y = (x + k) \bmod 26$

Odszyfrowanie odbywa się według wzoru: $x = (y - k) \bmod 26$

2.1.2. Przykład działania

Założmy, że szyfrujemy słowo KOT dla klucza $k = 3$.

| Litera jawna | Pozycja | Litera zaszyfrowana |
|--------------|---------|---------------------|
| K | 10 | N |
| O | 14 | R |
| T | 19 | W |

Po zaszyfrowaniu:

KOT \rightarrow NRW KOT \rightarrow NRW Aby odszyfrować tekst, każdą literę przesuwamy o k pozycji wstecz, otrzymując pierwotny tekst jawny.

2.2. Podsumowanie

Szyfr Cezara jest wyjątkowo prosty w użyciu i implementacji, dlatego często wykorzystywany jest jako przykład w nauczaniu podstaw kryptografii. Ze względu na ograniczoną liczbę możliwych kluczy, nie jest stosowany w praktycznych rozwiązaniach wymagających wysokiego poziomu bezpieczeństwa.

2.3. Running Key Cipher

Running Key Cipher (szyfr tekstu kluczowego) to odmiana szyfru Vigen'ere'a, gdzie zamiast powtarzającego się słowa kluczowego do szyfrowania wykorzystuje się cały wyrazisty tekst (tzw. "klucz bieżący"), np. fragment książki, artykułu czy innego dłuższego tekstu. Ten typ szyfru jest rodzajem szyfru polialfabetycznego, co oznacza, że każda litera wiadomości jest szyfrowana za pomocą odpowiedniej litery klucza według indywidualnego przesunięcia dla każdej pozycji.

2.3.1. Zasada działania

Niech:

- P – litera tekstu jawnego (plaintext),
- K – litera klucza (key),
- C – litera tekstu zaszyfrowanego (ciphertext).

Proces szyfrowania możemy zapisać wzorem:

$$C_i = (P_i + K_i) \bmod 26$$

Odszyfrowanie:

$$P_i = (C_i - K_i + 26) \bmod 26$$

Przykład:

Jeśli chcemy zaszyfrować słowo KOT kluczem DOM, przesuwamy każdą literę tekstu o liczbę pozycji odpowiadającą literze klucza na tej samej pozycji.

| Litera jawna | Litera klucza | Wynik | Wynik (litera) |
|--------------|---------------|-------|----------------|
| K (10) | D (3) | 13 | N |
| O (14) | O (14) | 2 | C |
| T (19) | M (12) | 5 | F |

KOT przy kluczu DOM zostanie zaszyfrowane jako NCF.

2.3.2. Podsumowanie

Running Key Cipher jest znacznie trudniejszy do złamania niż prosty szyfr Cezara, zwłaszcza jeśli klucz jest wystarczająco długi i niepowtarzalny. Jego bezpieczeństwo zależy jednak od tajności i jakości klucza (im bardziej losowy, tym lepiej).

2.4. Szyfr Beauforta

Szyfr Beauforta stanowi kolejną odmianę szyfru polialfabetycznego, podobnie jak Vigen'ere, lecz stosuje inny wzór szyfrowania. Klucz jest powtarzany lub wydłużany do długości tekstu jawnego, a proces szyfrowania przebiega według poniższego równania:

2.4.1. Zasada działania

Niech:

- P – pozycja litery tekstu jawnego w alfabecie,
- K – pozycja litery klucza w alfabecie,
- C – pozycja litery tekstu zaszyfrowanego.

Szyfrowanie:

$$C_i = (K_i - P_i + 26) \bmod 26$$

Odszyfrowanie algorytmu odbywa się tym samym wzorem, dzięki jego wzajemności: zamieniając rolę tekstu jawnego i zaszyfrowanego oraz klucza otrzymujemy pierwotny tekst.

Przykład:

Dla tekstu jawnego KOT i klucza DOM:

| Litera jawna | Litera klucza | Wynik | Litera szyfru |
|--------------|---------------|-------|---------------|
| K (10) | D (3) | 19 | T |
| O (14) | O (14) | 0 | A |
| T (19) | M (12) | 19 | T |

Szyfruje się więc: $KOT \rightarrow TAT$ (dla klucza DOM).

2.4.2. Podsumowanie

Szyfr Beauforta, dzięki specyficznej zasadzie działania, nadaje się zarówno do szyfrowania, jak i deszyfrowania przy tym samym wzorze matematycznym. Był używany m.in. w maszynach szyfrujących w XIX i XX wieku. Dobrą praktyką jest stosowanie długiego, niepowtarzalnego klucza dla zapewnienia bezpieczeństwa.

3. Technologie

Projekt został zrealizowany z wykorzystaniem języka programowania Python, który cechuje się czytelną składnią, dużą liczbą dostępnych bibliotek oraz szerokim wsparciem społeczności. Do implementacji graficznego interfejsu użytkownika wykorzystano bibliotekę **Tkinter**.

3.1. Python

Python to uniwersalny język programowania wysokiego poziomu, szeroko stosowany zarówno w aplikacjach webowych, jak i przy implementacji skryptów, narzędzi oraz aplikacji desktopowych. Jego zaletą jest duża liczba dostępnych bibliotek i frameworków, co pozwala na szybkie prototypowanie oraz implementację złożonych rozwiązań programistycznych.

3.2. Tkinter

Tkinter to standardowa biblioteka Pythona służąca do tworzenia graficznych interfejsów użytkownika (GUI). Udostępnia zestaw komponentów (tzw. widgetów), takich jak okna dialogowe, przyciski, pola tekstowe, rozwijane listy czy kontrolki wyboru. Tkinter umożliwia szybkie tworzenie aplikacji okienkowych bez konieczności instalowania dodatkowych zewnętrznych zależności — stanowi integralną część standardowej dystrybucji Pythona.

Podstawowe cechy Tkinter:

- Intuicyjna struktura widgetów umożliwiająca szybkie projektowanie okien aplikacji,
- Możliwość obsługi zdarzeń (np. kliknięcia przycisku, wprowadzenia tekstu),
- Łatwość integracji z innymi modułami Pythona oraz obsługi plików, operacji na danych czy funkcji kryptograficznych,
- Dobra dokumentacja i przykłady dostępne w materiałach społeczności Pythona.

3.3. System kontroli wersji – GitHub

Do zarządzania kodem źródłowym projektu wykorzystano system kontroli wersji **Git** oraz zewnętrzny serwis **GitHub**. Git umożliwia śledzenie historii zmian oraz pracę zespołową nad kodem aplikacji. GitHub ułatwia przechowywanie repozytorium w chmurze, współpracę, raportowanie błędów, prowadzenie changelogu oraz publikację dokumentacji.

Zalety korzystania z Git/GitHub:

- Bezpieczne przechowywanie i archiwizacja kodu źródłowego,
- Łatwe śledzenie postępu i zapisu kolejnych zmian,
- Możliwość przywrócenia wcześniejszych wersji projektu,
- Współpraca i synchronizacja pracy przy rozwoju kolejnych algorytmów kryptograficznych,
- Publiczna prezentacja projektu oraz dokumentacji.

Projekt oraz kod źródłowy dostępny jest publicznie na GitHub:

<https://github.com/Jaciesune/Cipher-Decipher>

4. Architektura oraz wygląd aplikacji

Aplikacja została zbudowana według obiektowego podejścia, które ułatwia przyszłą rozbudowę o kolejne algorytmy szyfrowania. Strukturę programu oparto na wydzieleniu klasy bazowej i klas dla konkretnych algorytmów.

4.1. Architektura techniczna

Podstawę aplikacji tworzą następujące klasy:

- **CipherAlgorithm** – klasa bazowa definiująca abstrakcyjne metody `encrypt` oraz `decrypt`. Każda nowa klasa szyfru musi je zaimplementować. Jest to zgodne z zasadą programowania w interfejs.
- **CaesarCipher** – realizuje szyfr Cezara, z parametrem przesunięcia przekazywanym w konstruktorze. Pozwala na przekształcenie każdej litery tekstu zgodnie z algorytmem podanym matematycznym wzorem:

$$y = (x + k) \bmod 26$$

dla szyfrowania oraz

$$x = (y - k) \bmod 26$$

dla odszyfrowania.

- **ReverseCipher** – implementuje prosty mechanizm odwrócenia kolejności znaków, umożliwiając zarówno szyfrowanie jak i deszyfrowanie przez tę samą operację (symetria).
- **EncryptionApp** – główna klasa dziedzicząca po `tk.Tk` z biblioteki `Tkinter`. Odpowiada za cały interfejs użytkownika, obsługę przycisków, integrację z klasami algorytmów oraz wczytywanie/zapisywanie plików tekstowych.

Kod został zaprojektowany tak, by obsługa nowych algorytmów wymagała jedynie stworzenia odpowiedniej klasy dziedziczącej po `CipherAlgorithm` i jej rejestracji w interfejsie (np. przez dodanie do listy wyboru algorytmów).

4.1.1. Przykładowe fragmenty kodu

Poniżej ilustracja definicji klasy bazowej i przykładowego algorytmu (skrótowa do fragmentu):

```
1 class CipherAlgorithm:
2     def encrypt(self, text):
3         raise NotImplementedError
4     def decrypt(self, text):
5         raise NotImplementedError
6
7 class CaesarCipher(CipherAlgorithm):
8     def __init__(self, shift=3):
9         self.shift = shift
10    def encrypt(self, text):
11        result = ''
12        for char in text:
13            if char.isalpha():
14                stay_in_alphabet = ord('A') if char.isupper() else
ord('a')
15                result += chr((ord(char) - stay_in_alphabet + self.
shift) % 26 + stay_in_alphabet)
16            else:
17                result += char
18        return result
```

Listing 1. Fragment kodu – klasa bazowa i szyfr Cezara

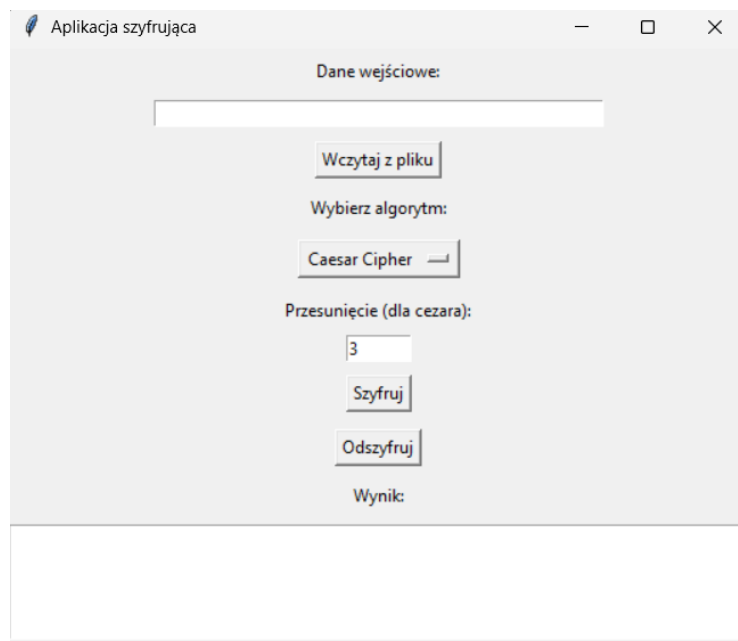
Do obsługi GUI wykorzystano standardową bibliotekę **Tkinter** – jest to rozbudowany moduł Pythona, pozwalający na szybkie tworzenie aplikacji okienkowych z gotowymi komponentami (widgetami). Dodatkowo, do wyboru i zapisu plików używane są `filedialog` oraz `messagebox`.

4.2. Wygląd aplikacji

Aplikacja posiada przejrzysty, minimalistyczny interfejs graficzny, zbudowany z komponentów Tkinter:

- Pole tekstowe do wpisania lub wczytania tekstu,
- Lista rozwijana do wyboru algorytmu,
- Pole do wpisania przesunięcia (dla szyfru Cezara),
- Przyciski: wczytania pliku, szyfrowania, odszyfrowania oraz zapisu wyniku,

- Pole wynikowe umożliwiające zaznaczenie i kopiowanie tekstu.



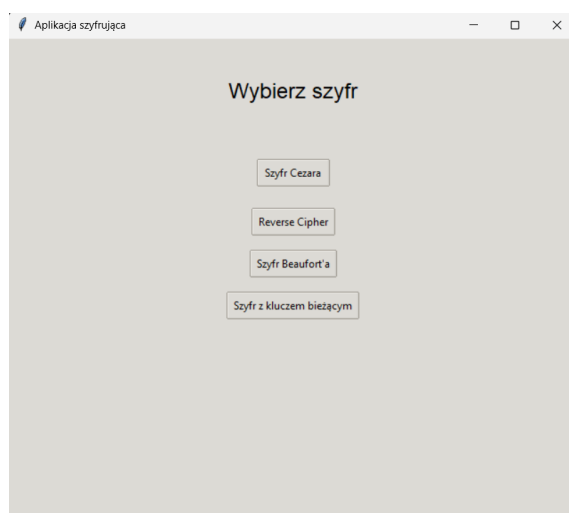
Rys. 4.1. Widok głównego okna aplikacji szyfrującej w Tkinter.

Architektura programu oraz wygląd interfejsu umożliwiają łatwe rozszerzanie projektu i gwarantują prostą obsługę dla użytkownika końcowego.

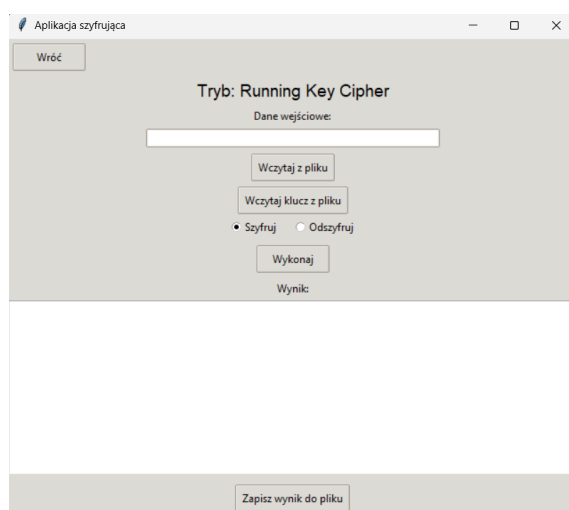
4.3. Aktualizacje wyglądu

14.10.2025

Wygląd został zaktualizowany na potrzebę uproszczenia budowy kolejnych algorytmów oraz przejrzystości wyglądu aplikacji.



Rys. 4.2. Zaktualizowany wygląd aplikacji w menu wyboru algorytmu szyfrowania



Rys. 4.3. Zaktualizowany wygląd aplikacji po wyborze jednej z opcji na rysunku 4.2

5. Changelog

- **13.10.2025** – Rozpoczęcie prac nad aplikacją szyfrującą w języku Python z wykorzystaniem biblioteki Tkinter do tworzenia interfejsu graficznego. Initial commit oraz wstępna konfiguracja repozytorium.
- **13.10.2025** – Dodano możliwość szyfrowania oraz odszyfrowywania tekstu przy użyciu szyfru Cezara (szyfr podstawieniowy z konfigurowalnym przesunięciem) oraz dodatkowy prosty algorytm szyfrujący (bonus).
- **13.10.2025** – Zmieniono nazwę głównego pliku projektu (`main.py`).
- **14.10.2025** – Dodano implementację szyfru odwracającego ciąg znaków (pierwszy znak staje się ostatnim, drugi przedostatnim itd.).
- **14.10.2025** – Wprowadzono separację wyboru algorytmu szyfrującego przy użyciu zakładek w interfejsie aplikacji.
- **14.10.2025** – Dodano nowy algorytm szyfrujący: **Beaufort cipher**.
- **14.10.2025** – Dodano nowy algorytm szyfrujący: **Running Key Cipher**; zaktualizowano pliki testowe Beauforta.
- **14.10.2025** – Wprowadzono funkcjonalność wczytywania tekstu z pliku oraz zapisu wyników szyfrowania/deszyfrowania do pliku.

Spis rysunków

| | |
|---|----|
| 4.1. Widok głównego okna aplikacji szyfrującej w Tkinter. | 11 |
| 4.2. Zaktualizowany wygląd aplikacji w menu wyboru algorytmu szyfrowania | 12 |
| 4.3. Zaktualizowany wygląd aplikacji po wyborze jednej z opcji na rysunku | |
| 4.2 | 12 |

Spis tabel

Spis listingów

1. Fragment kodu – klasa bazowa i szyfr Cezara 10