AKADEMIA NAUK STOSOWANYCH
W NOWYM SĄCZU

KRYPTOGRAFIA I TEORIA KODÓW

SPRAWOZDANIE

Autor: **Marcel Kasprzycki**
Rok: **4 IS(s) 2024/25**
Grupa: **P2**

NOWY SĄCZ – 15 GRUDNIA 2024

Spis treści

1 Wprowadzenie **4**

2 Kryptografia **5**

2.1 Szyfr Cezara . . . 5

2.2 Szyfr Transpozycyjny . . . 5

2.3 DES . . . 5

2.3.1 CBC . . . 5

2.3.2 CFB . . . 5

2.4 RSA . . . 6

2.5 Protokół Diffiego-Hellmana . . . 6

2.6 Certyfikaty . . . 6

2.7 Podpis cyfrowy . . . 6

2.7.1 SHA-256 . . . 6

2.8 HMAC . . . 7

3 Implementacja **8**

3.1 Szyfr Cezara . . . 8

3.2 Szyfr Transpozycyjny . . . 9

3.3 CBC . . . 10

3.4 CFB . . . 11

3.5 RSA . . . 12

3.6 Certyfikaty . . . 13

3.7 Podpis Cyfrowy . . . 14

3.8 HMAC . . . 15

4 Podsumowanie **16**

1 Wprowadzenie

Kryptografia jest jedną z ważnych gałęzi nauki, a jej celem jest zapewnienie poufności, bezpieczeństwa oraz weryfikacji autentyczności danych. Jej historia sięga czasów starożytnych, kiedy stosowano podstawowe metody szyfrowania, takie jak szyfry transpozycyjne czy szyfr Cezara. W dzisiejszych czasach kryptografia wykorzystuje znacznie bardziej zaawansowane modele matematyczne i algorytmy komputerowe, które pozwalają na szyfrowanie danych z zachowaniem poziomu bezpieczeństwa. Stanowi ona główny fundament ochrony informacji w takich dziedzinach, jak handel internetowy, systemy bankowe czy ochrona danych osobowych.

Celem niniejszej pracy jest analiza oraz prezentacja aplikacji demonstrującej różne techniki szyfrowania i deszyfrowania danych. Przedstawione zostały zarówno najprostsze metody, takie jak szyfr Cezara, jak i bardziej zaawansowane techniki, w tym RSA.

Aplikacja została opracowana w oparciu o język programowania Python, z wykorzystaniem bibliotek PySide6, OpenSSL oraz cryptography, które zapewniają wysoką wydajność i bezpieczeństwo przetwarzania danych.

2 Kryptografia

2.1 Szyfr Cezara

Szyfr Cezara, znany również jako szyfr przesuwający, jest jednym z najprostszych sposobów szyfrowania. Jego nazwa pochodzi od Juliusza Cezara, który prawdopodobnie wykorzystywał go do komunikacji. Działanie szyfru jest bardzo proste i polega na przesunięciu każdej litery o **n** miejsc w alfabecie. Na przykład, jeśli **n** wynosi 6, to zaszyfrowane zdanie „Ala ma kota kot ma ale" będzie wyglądało następująco: „Grg sg quzg quz sg Grk".

2.2 Szyfr Transpozycyjny

Szyfr transpozycyjny jest rodzajem szyfru przestawieniowego, który polega na przekształceniu liter tekstu jawnego poprzez wykorzystanie tabeli lub figury geometrycznej, a następnie odczytaniu tekstu według określonego wzorca. Przykładem może być wpisanie tekstu w tabelę i jego odczytanie kolumnami. Na przykład dla hasła „Ala ma kota kot ma Ale" wynik może wyglądać tak: „AMOKML LATOAE A AT K A"

2.3 DES

DES, czyli Data Encryption Standard, to algorytm szyfrowania blokowego opracowany przez IBM, który działa na blokach danych o długości 64 bitów. Proces szyfrowania rozpoczyna się od permutacji początkowej, która przestawia bity bloku wejściowego według ustalonego schematu. Następnie blok dzielony jest na dwie równe części – lewą i prawą. W trakcie każdej z 16 rund prawa część jest rozszerzana do 48 bitów, a wynik poddawany operacji XOR z podkluczem wygenerowanym z klucza głównego. Wynik tej operacji przechodzi przez S-bloki, które zmniejszają długość danych do 32 bitów, a następnie jest poddawany dodatkowej permutacji. Kolejnym krokiem jest zamiana lewej i prawej części, co wprowadza dodatkowe bezpieczeństwo. Po zakończeniu wszystkich rund dwie części są łączone i poddawane permutacji końcowej, która odwraca działanie permutacji początkowej, tworząc zaszyfrowany blok danych. Proces deszyfrowania przebiega w analogiczny sposób, z odwrotną kolejnością użycia podkluczy.

2.3.1 CBC

CBC, czyli Cipher Block Chaining, to jeden z trybów działania szyfru DES, w którym każdy blok tekstu jawnego jest dodatkowo XORowany z poprzednim zaszyfrowanym blokiem. Wektor inicjalizacyjny (IV) jest używany do szyfrowania pierwszego bloku, co wprowadza losowość i zwiększa bezpieczeństwo całej struktury.

2.3.2 CFB

CFB, czyli Cipher Feedback, to tryb działania DES, który zamiast operować na blokach danych, szyfruje dane w sposób strumieniowy, czyli ciągły. W tym trybie poprzedni blok szyfrogramu jest szyfrowany, a wynik XORowany z kolejnym blokiem tekstu jawnego. Dzięki temu możliwe jest szyfrowanie danych o zmiennej długości, a także ich przesyłanie w czasie rzeczywistym.

2.4 RSA

RSA a dokładniej Rivest–Shamir–Adleman to algorytm asymetryczny, który opiera swoje działanie na dużych liczbach pierwszych. Wykorzystuje on parę kluczy pierwszy nazywany publiczny służy do szyfrowania wiadomości natomiast drugi zwany prywatnym działa do odszyfrowywania zaszyfrowanej wiadomości.Proces szyfrowania i deszyfracji opiera zsie na konkretnych działaniach matematycznych.

2.5 Protokół Diffiego-Hellmana

Protokół Diffiego-Hellmana to metoda, która umożliwia dwóm stronom bezpieczną i tajną wymianę informacji, takich jak klucze prywatny do RSA,nawet jeśli komunikacja odbywa się przez publiczny kanał. Protokół opiera się na trudności rozwiązania problemu logarytmu dyskretnego, co zapewnia jego bezpieczeństwo.

Sposób działania opiera sie na tym że obie strony wybierają wspólną liczbę podstawową oraz moduł, a następnie każda ze stron generuje swój klucz prywatny. Na podstawie tych wartości obliczają tzw. klucze publiczne, które wymieniają między sobą. Dzięki swoim kluczom prywatnym oraz otrzymanym kluczom publicznym obie strony są w stanie obliczyć wspólny klucz.

2.6 Certyfikaty

Certyfikaty cyfrowe to elektroniczne dokumenty oparte na kluczu publicznym, które służą do potwierdzania tożsamości podmiotów, takich jak osoby, organizacje czy firmy. Są one wystawiane przez zaufane jednostki trzecie, zwane urzędami certyfikacji. Certyfikaty funkcjonują w ramach tzw. łańcuchów certyfikatów, czyli hierarchii certyfikatów służących do weryfikacji autentyczności podmiotu. Przerwanie ważności jednego z certyfikatów w łańcuchu powoduje unieważnienie całego łańcucha.

2.7 Podpis cyfrowy

Podpis cyfrowy to metoda uwierzytelniania dokumentu elektronicznego, która zapewnia, że dokument pochodzi od określonego nadawcy i nie został zmieniony podczas przesyłania. Działa poprzez wygenerowanie unikalnego skrótu z dokumentu, który następnie jest szyfrowany kluczem prywatnym nadawcy. Odbiorca, używając klucza publicznego nadawcy, odszyfrowuje skrót i porównuje go z nowo wygenerowanym skrótem z otrzymanego dokumentu. W przypadku jakiejkolwiek modyfikacji dokumentu po jego podpisaniu, skróty nie będą się zgadzać, co umożliwia wykrycie nieautoryzowanych zmian.

2.7.1 SHA-256

SHA-256 to jeden z najpopularniejszych algorytmów kryptograficznych wykorzystywanych w technologii certyfikatów cyfrowych. Jego głównym zadaniem jest przekształcenie danych wejściowych, takich jak tekst w unikalny skrót o stałej długości 256 bitów, co w systemie szesnastkowym odpowiada 64 znakom. Działa on w kilku krokach na początku dzieli dane na bloki o wielkości 512 bitów, jeśli są krótrze dodaje on bity aby osiągnoąć odpowiednią długość. Nastepnie każdy blok przechodzi przez szereg operacji matematycznych, takich jak przesunięcia bitowe, dodawanie modulo czy funkcje logiczne, które są zaprojektowane tak, aby wynikowy skrót był trudny do przewidzenia i zależał od każdej zmiany w danych wejściowych.Po przetworzeniu wszystkich bloków algorytm generuje skrót pełniący rolę unikalnego identyfikatora danych.

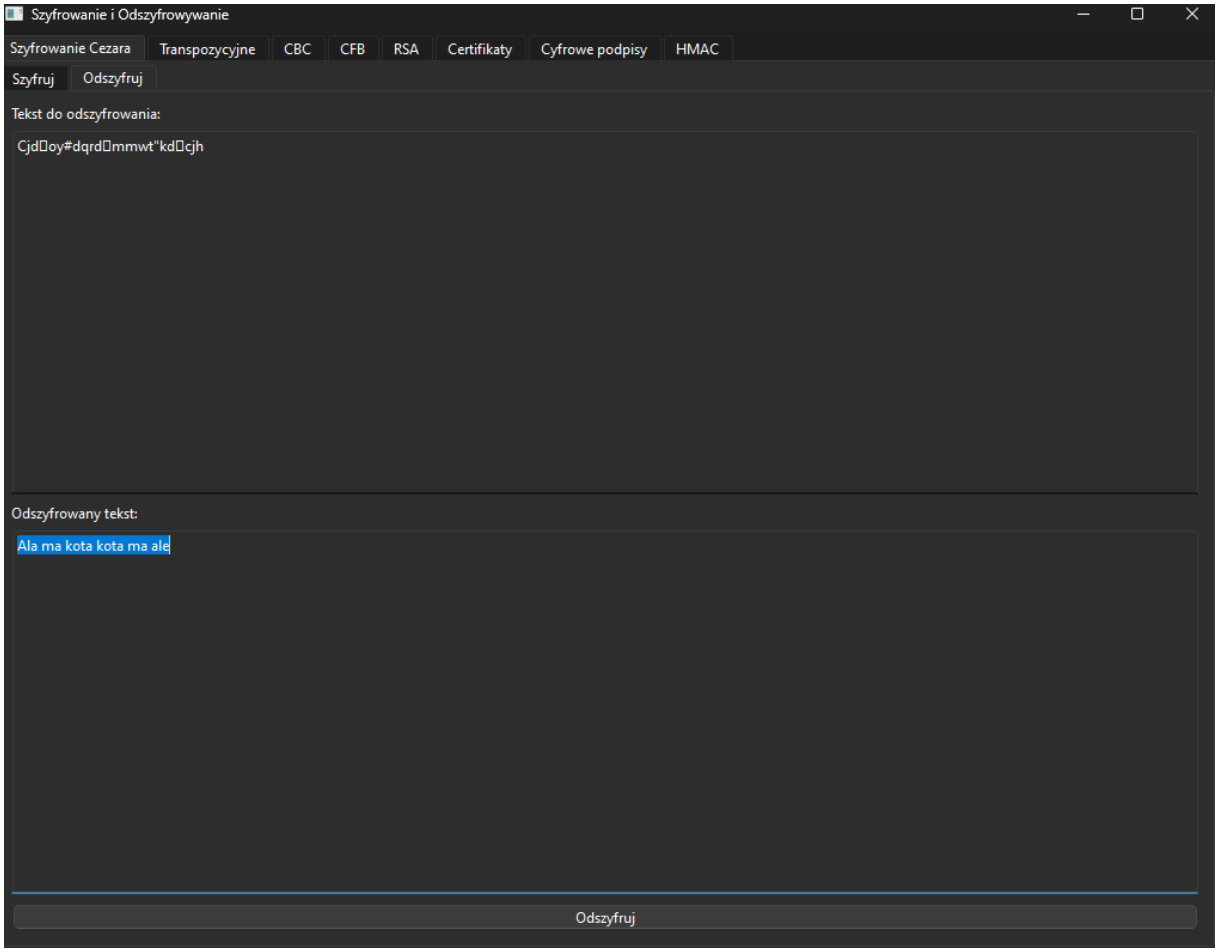
2.8 HMAC

HMAC to jedna z metod kryptograficznego uwierzytelniania wiadomości, oparta na funkcjach skrótu. Umożliwia ona sprawdzenie, czy wiadomość została przesłana w bezpieczny sposób. Proces generowania kodu HMAC można przedstawić w kilku krokach. Na początku klucz tajny jest łączony z tekstem jawnym. Następnie wynik poprzedniego kroku jest przetwarzany przez funkcję skrótu, tworząc tymczasowy skrót. W kolejnym kroku utworzony skrót jest łączony z kluczem i ponownie przetwarzany przy pomocy funkcji skrótu. Na końcu utworzony skrót jest dołączany do wiadomości jako kod HMAC.

3 Implementacja

3.1 Szyfr Cezara

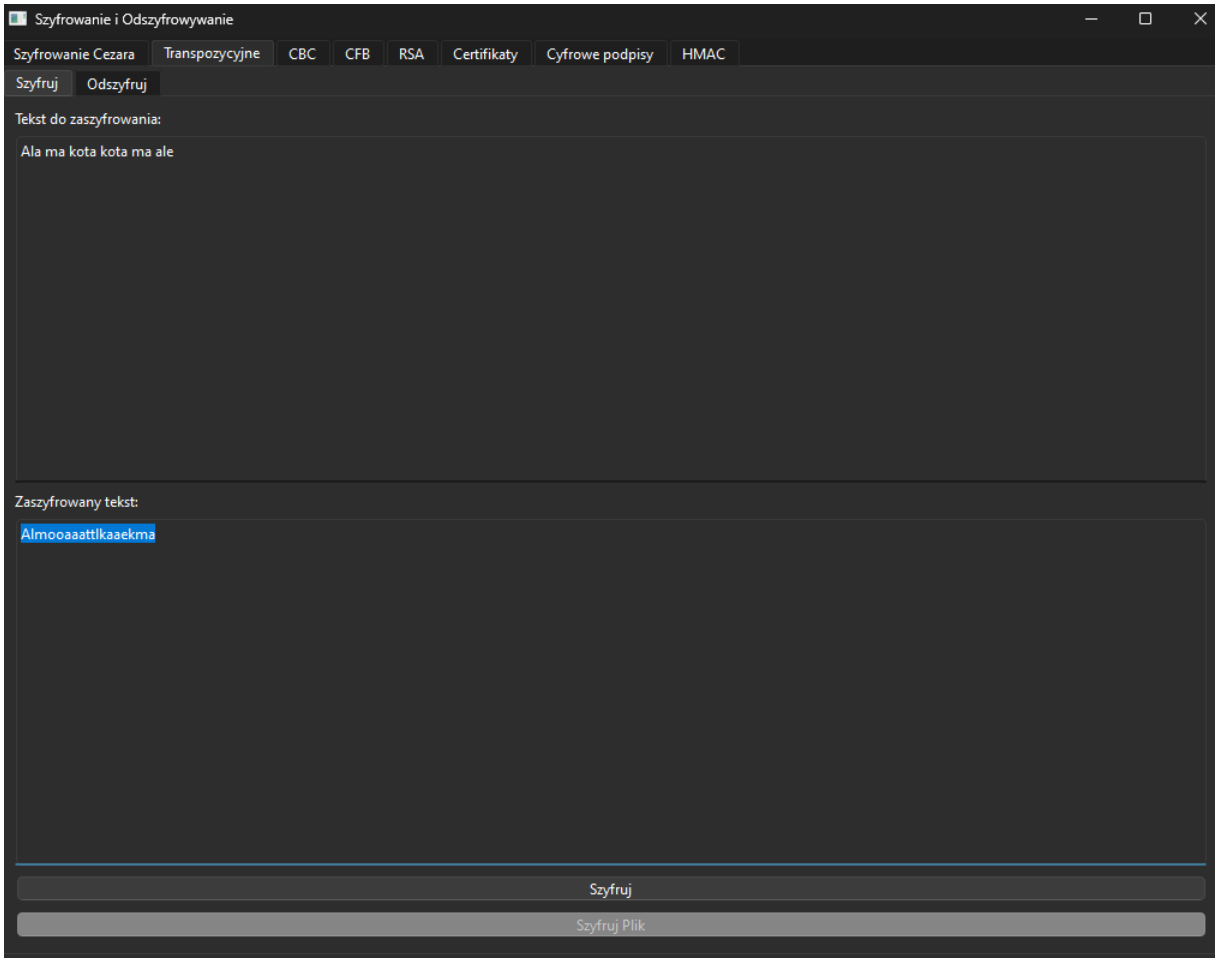
W implementacji szyfru Cezara wykorzystałem zmodyfikowana wersję algorytmu, która różni się od jego klasycznej formy polegającej na przesuwaniu każdej litery o stałą wartość. Zamiast tego zastosowano bardziej zaawansowaną metodę, opartą na naprzemiennym przesuwaniu liter w przód i w tył zgodnie z kluczem 2237.



Rysunek 1: Wygląd podstrony od odszyfrowywania szyfru cezara

3.2 Szyfr Transpozycyjny

W implementacji szyfru transpozycyjnego wykorzystanłem strukturę trójkąta równobocznego, w którym dane wejściowe są wpisywane wierszami zgodnie z kształtem trójkąta, a następnie odczytywane kolumnami od góry do dołu, co generuje zaszyfrowany tekst bez spacji. Na początku z wiadomości usuwane są spacje, a następnie obliczana jest liczba wierszy trójkąta przy użyciu wzoru matematycznego dla sumy ciągu arytmetycznego. Każdy wiersz trójkąta zawiera o jedną literę więcej niż poprzedni, a brakujące znaki w ostatnim wierszu są uzupełniane spacjami. Po wypełnieniu trójkąta znaki są odczytywane kolumnami,



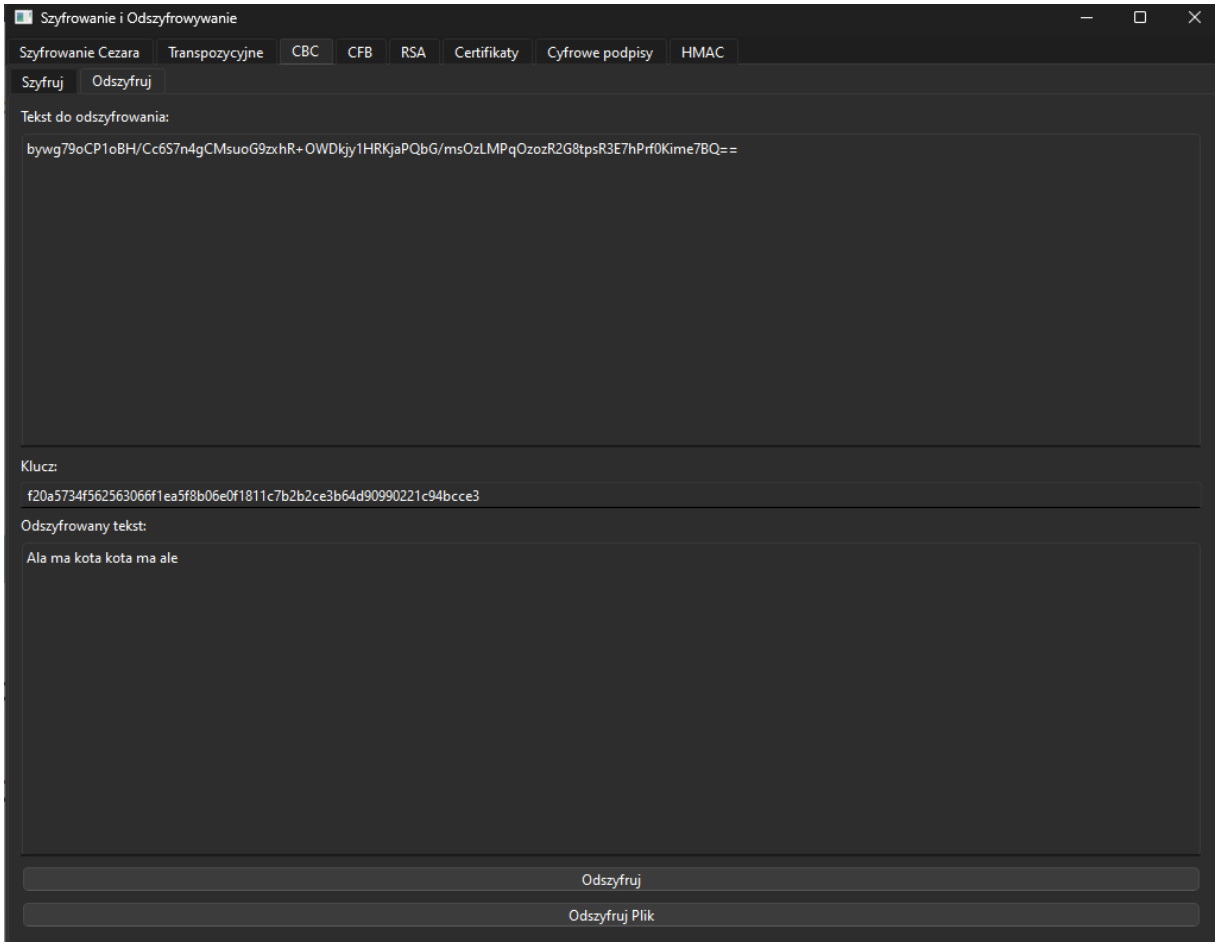
Rysunek 2: Wygląd podstrony dla szyfru transpozycyjnego

```python
def create_triangle_structure(message):
    n = int(math.ceil((-1 + math.sqrt(1 + 8 * len(message))) / 2))
    triangle = []
    index = 0
    for row in range(1, n + 1):
        if index + row <= len(message):
            triangle.append(list(message[index:index + row]))
        else:
            triangle.append(list(message[index:] + ' ' * (row - (len(message) - index))))
        index += row
    return triangle
```

Listing 1: Funkcja tworząca strukture trójkąta

3.3 CBC

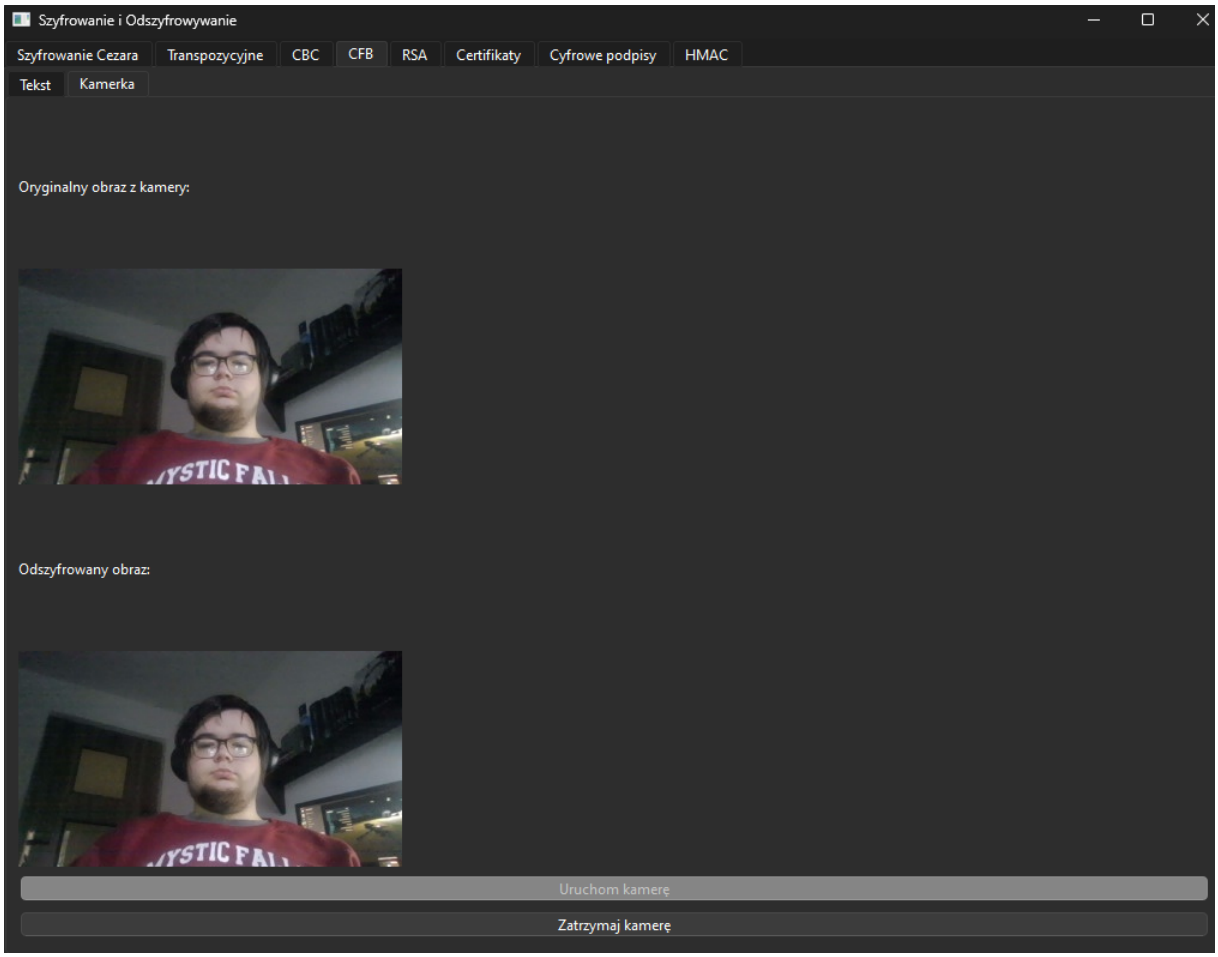
Do implementacji szyfrowania CBC umożliwiającej szyfrowanie zarówno tekstów jak i plików wykorzystano bibliotekę PyCryptodome oraz algorytm AES w trybie CBC. Dane wejściowe są konwertowane na format bajtów i dopełniane do rozmiaru bloku. Proces szyfrowania obejmuje generowanie losowych parametrów czyli soli oraz wektora inicjującego. Wynik szyfrowania, zawierający zaszyfrowane dane, sól i IV, jest kodowany w Base64, co umożliwia łatwą reprezentację tekstową i przechowywanie. Podczas deszyfrowania dane są dekodowane z Base64. Funkcja obsługuje także pliki, które są szyfrowane jako dane binarne z zastosowaniem tych samych zasad.



Rysunek 3: Wygląd podstrony dla CBC

3.4 CFB

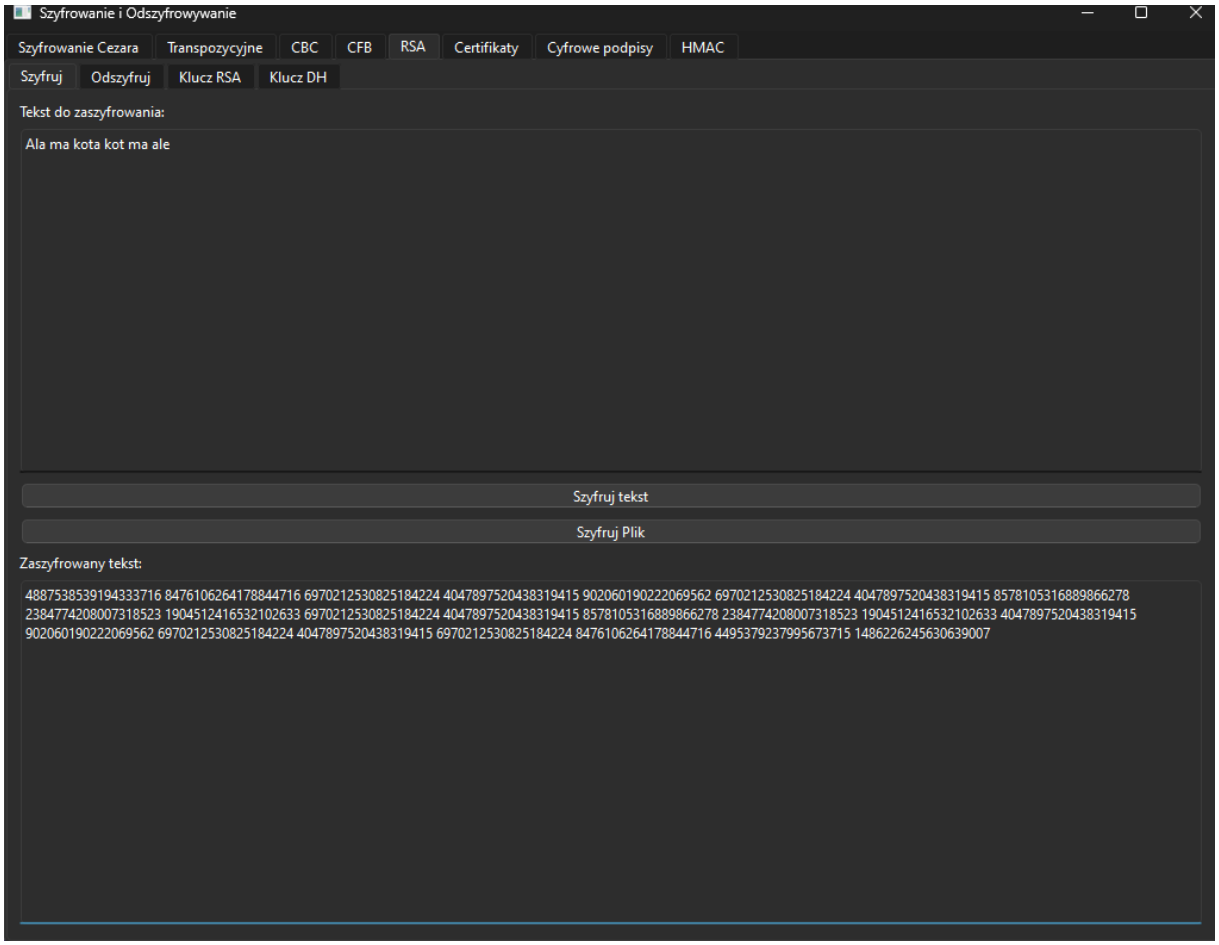
Do implementacji szyfrowania w trybie CFB (Cipher Feedback Mode) wykorzystano bibliotekę PyCryptodome do obsługi algorytmu AES oraz OpenCV do przetwarzania obrazu z kamery. Tryb CFB, w przeciwieństwie do CBC, przetwarza dane w mniejszych fragmentach, umożliwiając szyfrowanie strumieni w czasie rzeczywistym bez konieczności ich dopełniania. Obraz z kamery internetowej jest odczytywany klatka po klatce, przekształcany na dane binarne, a następnie szyfrowany i przesyłany dalej. Użytkownik może również szyfrować dane wprowadzone w polu tekstowym. Szyfrowanie w trybie CFB jak w CBC wymaga klucza 16,24 lub 32 bajtowego które można wygenerować w aplikacji.



Rysunek 4: Podgląd działania szyfrowania kamerki za pomocą CFB

3.5 RSA

Implementacja algorytmu RSA składa się z kilku kroków. Pierwszym z nich jest wygenerowanie kluczy Diffie-Hellmana, które będą służyły do zaszyfrowania klucza prywatnego. W procesie generowania liczb p i q można skorzystać z funkcji lub danych dostarczonych przez użytkownika. Następnie tworzona jest para kluczy RSA klucz publiczny, który zapisywany jest w zmiennej środowiskowej,i klucz prywatny, który przechowywany jest w pliku tekstowym. Po przygotowaniu kluczy możliwe jest szyfrowanie wiadomości. Do odszyfrowania wiadomości konieczne jest podanie klucza prywatnego. Dodatkowo algorytm umożliwia szyfrowanie i deszyfrowanie plików, co odbywa się w dwóch etapach najpierw plik jest kodowany w formacie BASE64, a następnie szyfrowany algorytmem RSA. Niestety, szybkośćtych operacjijest niska ze względu na działania tego algorytmu.



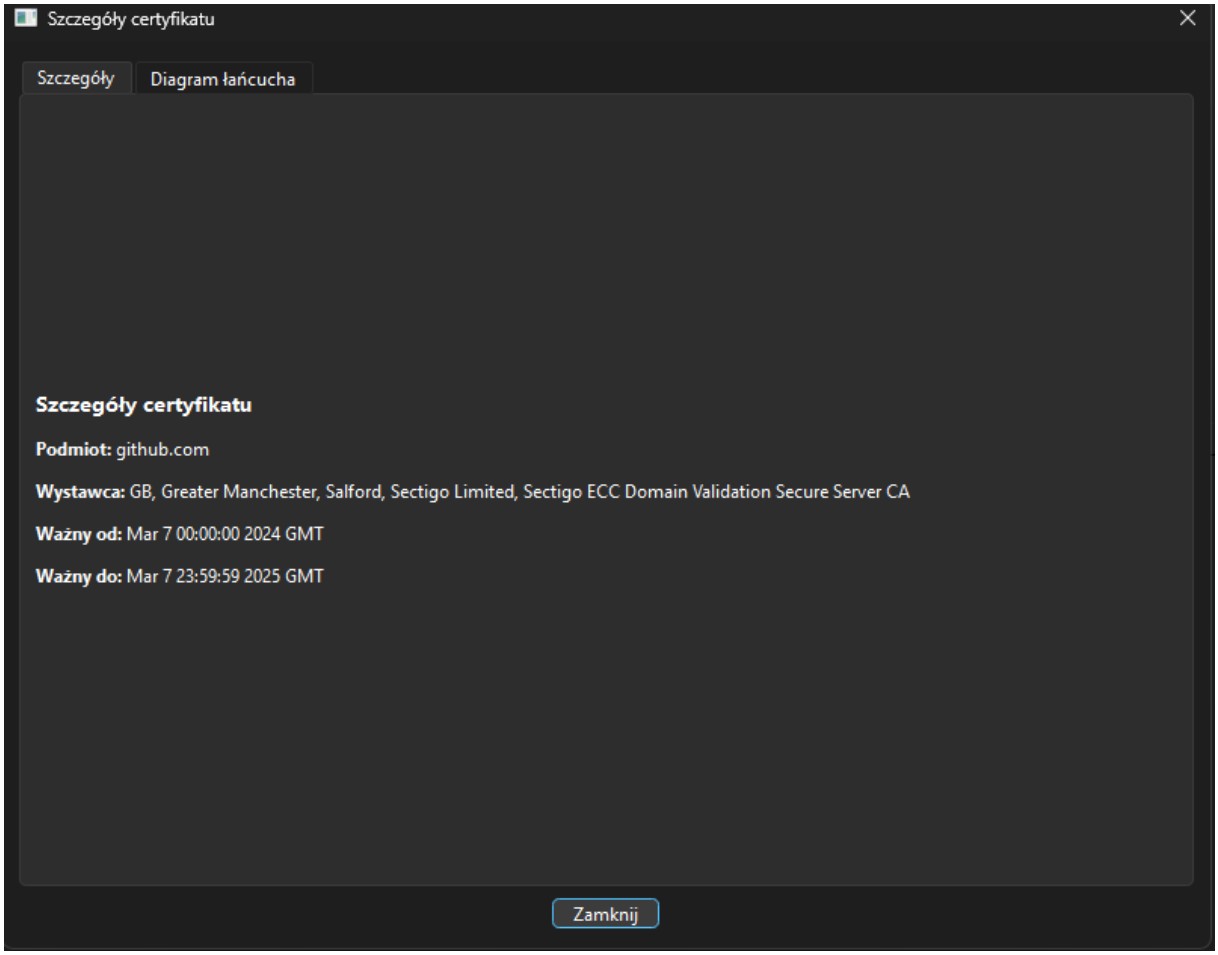
Rysunek 5: Wygląd podstrony do szyfrowania RSA

```python
def generate_rsa_keys(p, q):
    if p == q:
        raise ValueError("Liczby pierwsze p i q ąmusz ćby żróne.")
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537
    if gcd(e, phi) != 1:
        raise ValueError("e i phi nie ąs ęwzgldnie pierwsze.")
    d = mod_inverse(e, phi)
    return (e, n), (d, n)
```

Listing 2: Funkcja generująca parę kluczy RSA

3.6 Certyfikaty

Implementacja funkcji odpowiedzialnej za wyświetlanie certyfikatów oraz ich łańcuchów wykorzystuje biblioteki cryptography, urllib i OpenSSL. Proces rozpoczyna się od podania linku do strony, z której pobierane są certyfikaty. Następnie wyświetlane jest okno zawierające szczegółowe informacje o certyfikacie oraz graficzną reprezentację łańcucha certyfikatów, pokazującą jego strukturę i długość.



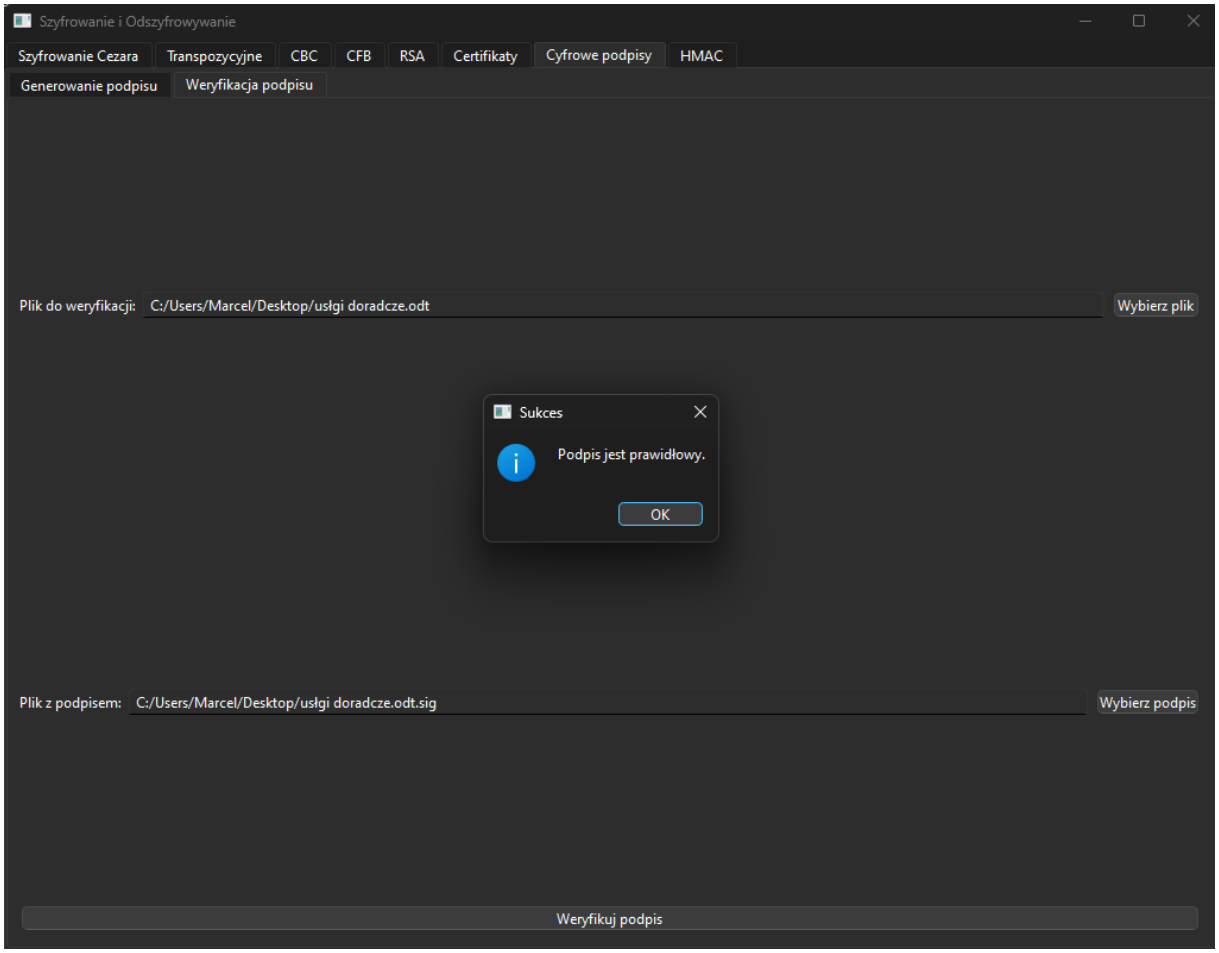
Rysunek 6: Enter Caption

```python
def generate_rsa_keys(p, q):
    if p == q:
        raise ValueError("Liczby pierwsze p i q ąmusz ćby żróne.")
    n = p * q
    phi = (p - 1) * (q - 1)
    e = 65537
    if gcd(e, phi) != 1:
        raise ValueError("e i phi nie ąs ęwzgldnie pierwsze.")
    d = mod_inverse(e, phi)
    return (e, n), (d, n)
```

Listing 3: Funkcja generująca parę kluczy RSA

3.7 Podpis Cyfrowy

Implementacja podpisu cyfrowego wykorzystuje bibliotekę cryptography, która umożliwia zarówno tworzenie, jak i weryfikację podpisów cyfrowych. Proces ten składa się z kilku etapów. Najpierw generowane są klucze, które zapisywane są w formacie PEM w podkatalogu ”klucze”. Klucz prywatny służy do podpisywania danych, natomiast klucz publiczny pozwala na weryfikację podpisu. Podpis cyfrowy generowany jest na podstawie wskazanego pliku z wykorzystaniem klucza. Zawartość pliku odczytywana jest jako ciąg bajtów, a następnie tworzony jest podpis przy użyciu algorytmu PSS i skrótu SHA-256. Podpis zapisywany jest w pliku z rozszerzeniem .sig. Do weryfikacji podpisu potrzebny jest plik z danymi, pliku podpisu oraz klucza publicznego.



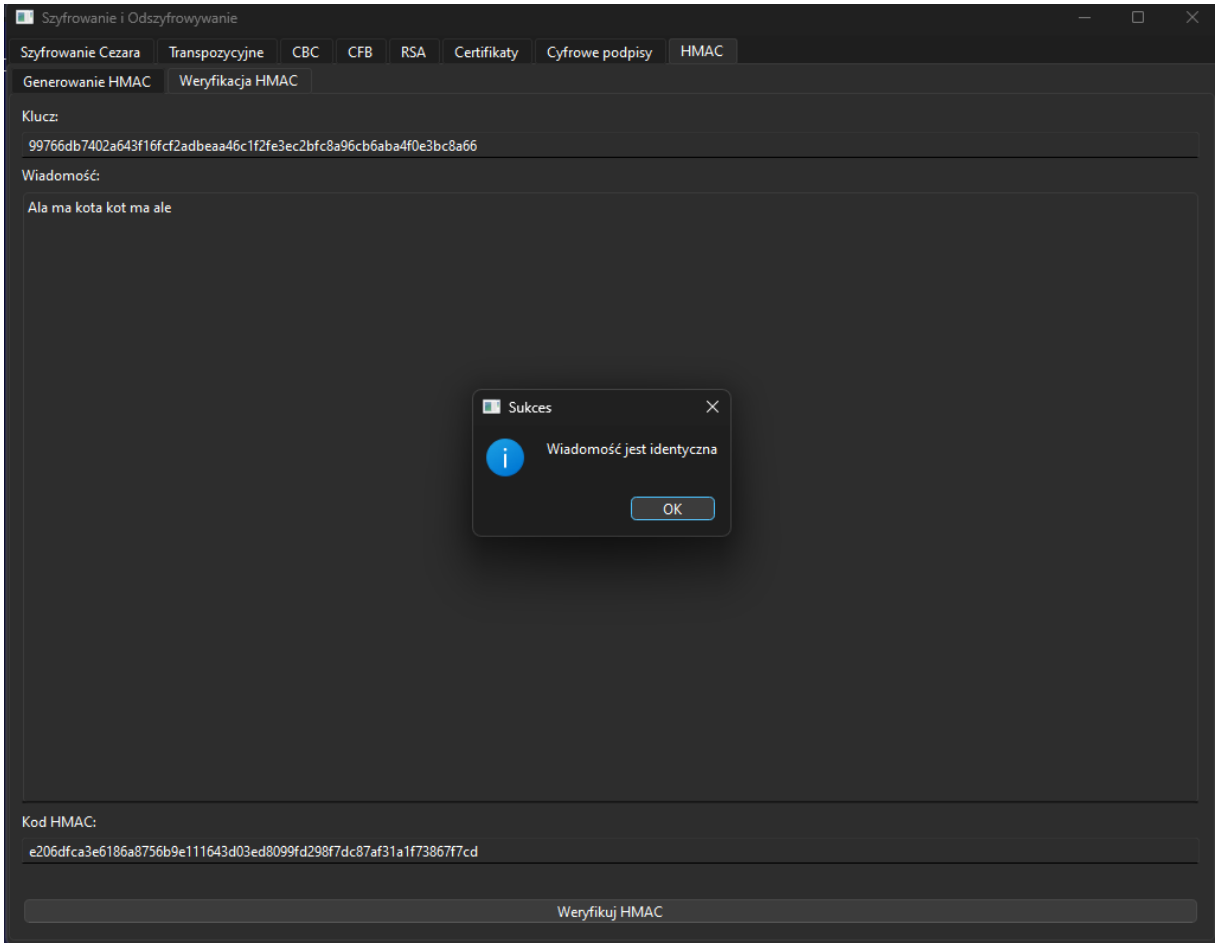
Rysunek 7: Wynik sprawdzania certyfikatu

```python
    def sign_file(file_path, private_key_file="private_key.pem"):
        with open(private_key_file, "rb") as key_file:
            private_key = serialization.load_pem_private_key(key_file.read(), password=None,)
        with open(file_path, "rb") as f:
        data = f.read()
        signature = private_key.sign(data,padding.PSS(mgf=padding.MGF1(hashes.SHA256()),
    salt_length=padding.PSS.MAX_LENGTH,), hashes.SHA256(),)
        signature_file = f"{file_path}.sig"
        with open(signature_file, "wb") as sig_file:
            sig_file.write(signature)
        return signature_file
```

Listing 4: Funkcja odpowiedzialna za tworzenie cerytyfikatu za pomocą SHA256

3.8 HMAC

Implementacja algorytmu HMAC w aplikacji działa wykorzystując wbudowaną w pythona biblioteki hmac oraz hashlib, które umożliwiają zapewnienie integralności i uwierzytelniania wiadomości. Funkcja szyfrująca generuje kod HMAC za pomocą podanej wcześniej wiadomości i kluczowi. Na początku weryfikuje poprawność danych wejściowych, a następnie konwertuje klucz i wiadomość na bajty za pomocą kodowania UTF-8. Wygenerowany w taki sposób HMAC jest zwracany w formie ciągu szesnastkowego. Funkcja sprawdzająca sprawdza zgondość podanego HMAC z wiadmością i kluczem.



Rysunek 8: Enter Caption

```python
def encrypt(message, key):
        try:
            if not message:
                raise ValueError("śćWiadomo nie żmoe ćby pusta.")
            if not key:
                raise ValueError("Klucz nie żmoe ćby pusty.")
            key_bytes = key.encode('utf-8')
            message_bytes = message.encode('utf-8')
            hmac_result = hmac.new(key_bytes, message_bytes, hashlib.sha256)
            return hmac_result.hexdigest()
        except Exception as e:
            raise Exception(f"łąBd podczas generowania HMAC: {e}")
```

Listing 5: Funkcja odpowiedzialna za działanie tworzenie kodu HMAC

4 Podsumowanie

Praca przedstawia zagadnienia kryptograficzne oraz ich implementację w aplikacji, umożliwiającej wykonywanie operacji takich jak szyfrowanie wiadomości i plików, weryfikacja certyfikatów oraz generowanie i sprawdzanie podpisów cyfrowych. Zastosowano szeroki wachlarz algorytmów, od podstawowych, takich jak szyfr Cezara czy transpozycyjny, po bardziej zaawansowane, w tym DES, algorytmy strumieniowe, RSA oraz HMAC. Pozwoliło to zaprezentować zarówno podstawy, jak i zaawansowane mechanizmy kryptograficzne. Dzięki wykorzystaniu biblioteki PySide6 opracowano prosty, intuicyjny interfejs graficzny, który usprawnia obsługę aplikacji i czyni ją przystępną zarówno dla początkujących, jak i zaawansowanych użytkowników. Interfejs umożliwia łatwe przeprowadzanie operacji kryptograficznych, wizualizację wyników oraz lepsze zrozumienie działania poszczególnych algorytmów. Projekt pełni rolę skutecznego narzędzia dydaktycznego, łącząc teorię z praktyką w przystępny sposób. Umożliwia eksperymentowanie z różnymi algorytmami kryptograficznymi, co wspiera proces nauki i ułatwia zrozumienie ich zastosowań w praktyce, takich jak ochrona danych, uwierzytelnianie czy zapewnienie poufności informacji. Aplikacja stanowi również solidną podstawę do dalszego rozwoju, co zwiększa jej potencjał edukacyjny i praktyczny.

Spis rysunków

1 Wygląd podstrony od odszyfrowywania szyfru cezara 8
2 Wygląd podstrony dla szyfru transpozycyjnego 9
3 Wygląd podstrony dla CBC 10
4 Podgląd działania szyfrowania kamerki za pomocą CFB 11
5 Wygląd podstrony do szyfrowania RSA 12
6 Enter Caption 13
7 Wynik sprawdzania certyfikatu 14
8 Enter Caption 15

Bibliografia

[1] *wikipedia*, UML: https://en.wikipedia.org/wiki/Data_Encryption_Standard, dostęp: 2024-12-14.

[2] *wikipedia*, UML: https://en.wikipedia.org/wiki/RSA_(cryptosystem), dostęp: 2024-12-14.

[3] *geeksforgeeks*, UML: https://www.geeksforgeeks.org/caesar-cipher-in-cryptography/, dostęp: 2024-12-14.

[4] *geeksforgeeks*, UML: https://www.geeksforgeeks.org/block-cipher-modes-of-operation/, dostęp: 2024-12-14.

[5] *geeksforgeeks*, UML: https://www.geeksforgeeks.org/implementation-diffie-hellman-algorithm/, dostęp: 2024-12-14.

[6] *ssl*, UML: https://www.ssl.com/pl/artykuł/co-to-jest-certyfikat-cyfrowy/, dostęp: 2024-12-14.

[7] *wikipedia*,UML: https://en.wikipedia.org/wiki/Digital_signature, dostęp: 2024-12-14.

[8] *howtointerview*, UML: https://howtointerview.pl/definicje/czym-jest-hash-based-message-authentication-c
11484/, dostęp: 2024-12-14.