Dzmitry Haurylkevich
Mariya Terechtchenko

RSA

Wprowadzenie:

Projekt przedstawiony w tej dokumentacji jest realizacją algorytmu szyfrowania RSA (Rivest-Shamir-Adleman) przy użyciu języka programowania Python. RSA jest jednym z najbardziej znanych algorytmów szyfrowania z kluczem publicznym opartym na matematycznym problemie faktoryzacji dużych liczb całkowitych. RSA wykorzystuje parę kluczy: klucz publiczny i klucz prywatny. Klucz publiczny służy do szyfrowania danych, a klucz prywatny do ich deszyfrowania.

Cele projektu:

Głównym celem projektu jest udostępnienie funkcjonalności umożliwiającej szyfrowanie i deszyfrowanie wiadomości przy użyciu algorytmu RSA. Projekt ma również na celu zapewnienie łatwego w użyciu interfejsu do obsługi szyfrowania i generowania kluczy, a także zapisywania i odczytywania kluczy z plików. Projekt został zaprojektowany z myślą o łatwości użytkowania i zrozumienia, tak aby każdy mógł z łatwością korzystać z jego funkcjonalności.

Podstawowe zasady działania:

W naszym projekcie, do szyfrowania danych z kluczem publicznym, używamy biblioteki kryptograficznej `cryptography`, która dostarcza narzędzia do generowania kluczy RSA, szyfrowania i deszyfrowania wiadomości.
Szyfrowanie kluczem publicznym: Projekt wykorzystuje algorytm RSA, który opiera się na parze kluczy - kluczu publicznym i kluczu prywatnym. Klucz publiczny służy do bezpiecznego szyfrowania danych, podczas gdy klucz prywatny jest używany do ich rozszyfrowywania. Działanie algorytmu zapewnia bezpieczną transmisję danych, ponieważ klucz publiczny może być udostępniany publicznie, natomiast klucz prywatny jest trzymany w tajemnicy przez właściciela.

Proces szyfrowania danych jest realizowany w funkcji `*encrypt(message, public_key)*`. Ta funkcja przyjmuje jako argumenty wiadomość do zaszyfrowania i klucz publiczny w formacie tekstowym. Wewnątrz funkcji, klucz publiczny jest importowany z podanego ciągu znaków `*public_key*` za pomocą metody `*RSA.import_key()*`. Następnie tworzony jest obiekt `*cipher*` z użyciem algorytmu `PKCS1_OAEP`, który jest oparty na kluczu otrzymanym z `*recipient_key*`.

Wiadomość jest kodowana do postaci bajtowej z użyciem kodowania UTF-8 (`*message.encode()*`) i następnie szyfrowana przy użyciu obiektu `*cipher*` *(`cipher.encrypt()`)*. Zaszyfrowana wiadomość jest reprezentowana jako ciąg znaków bajtowych.

Następnie, zaszyfrowana wiadomość jest przekształcana do formatu Base64 za pomocą funkcji `base64.b64encode()`. Ten krok umożliwia bezpieczne przechowywanie i przesyłanie zaszyfrowanej `*encrypt*` i może być wykorzystana do bezpiecznej transmisji lub przechowywania danych. Wiadomość jest zapisnywana w `cripto.txt`.
Możliwość szyfrowania wiadomości o maksymalnym ciągu 100 znaków.

Generowanie kluczy: Generowanie kluczy opiera się na złożonych obliczeniach matematycznych i zapewnia, że pary kluczy są unikalne dla każdego użytkownika. W projekcie została zaimplementowana funkcja *generate_key_pair(),* która generuje parę kluczy - klucz prywatny i klucz publiczny.
W funkcji *generate_key_pair()* najpierw generowany jest klucz o długości 2048 bitów przy użyciu algorytmu RSA za pomocą metody RSA.generate(2048).

Następnie wygenerowany klucz prywatny jest eksportowany do postaci tekstowej za pomocą *key.export_key()*, a klucz publiczny jest również eksportowany, ale w formie klucza publicznego za pomocą *key.publickey().export_key()*. Oba klucze są zapisywane do plików, gdzie klucz prywatny jest zapisywany w pliku "private_key.pem", a klucz publiczny w pliku "public_key.pem", przy użyciu funkcji *write_to_file(key, filename)*. Umożliwia to zapisywanie kluczy do późniejszego wykorzystania i wymiany między członkami systemu.

Na końcu funkcja zwraca wygenerowany klucz prywatny i klucz publiczny jako wynik. Te klucze mogą być wykorzystane w procesie szyfrowania i deszyfrowania danych przy użyciu algorytmu RSA.

Interfejs użytkownika: Projekt zapewnia graficzny interfejs użytkownika wykorzystujący bibliotekę "Tkinter". Ułatwia to korzystanie z programu i interakcję z funkcjami.

Instalacja:

Następujące zależności muszą być zainstalowane w celu pracy z kodem: pycryptodome: pip install pycryptodome; biblioteka «Tkinter».

Podręcznik użytkownika: opis funkcjonalności projektu i instrukcje dotyczące jego użytkowania.

1. „main.py” – plik główny

encrypted_message = encrypt(message, public_key) - szyfruje otrzymaną wiadomość, "public_key" reprezentuje klucz publiczny, który ma być użyty do szyfrowania.

decrypted_message = decrypt(encrypted_message, private_key) - deszyfruje zaszyfrowaną wiadomość, private_key reprezentuje klucz prywatny, który zostanie użyty do deszyfrowania.

private_key = read_to_file('private_key.pem') - odczytuje zawartość pliku "private_key.pem".

public_key = read_to_file('public_key.pem') - odczytuje zawartość pliku "public_key.pem".

new_key_button() - funkcja uruchamiana po naciśnięciu przycisku "Generuj nowe klucze", która generuje nową parę kluczy i zapisuje je w plikach "private_key.pem" i "public_key.pem".

exit_program() - funkcja uruchamiana po naciśnięciu przycisku " Zamknij", który zamyka aplikację.

window.mainloop() – uruchamia okno, czeka i przetwarza dane wejściowe użytkownika do momentu zamknięcia okna.

2. „cryptography.py” zawiera funkcje do pracy z kryptografią.

new_keys() - generuje nową parę kluczy (publiczny i prywatny) i przechowuje je w plikach "private_key.pem" i "public_key.pem".

generate_key_pair() - generuje nową parę kluczy (publiczny i prywatny).

encrypt(message, public_key) - szyfruje wiadomość przy użyciu klucza publicznego RSA.

decrypt(encrypted_message, private_key) - rozszyfrowuje zaszyfrowaną wiadomość przy użyciu klucza prywatnego

3. „key.py” zawiera funkcje do pracy z kluczami RSA, w tym odczytu i zapisu kluczy do plików.

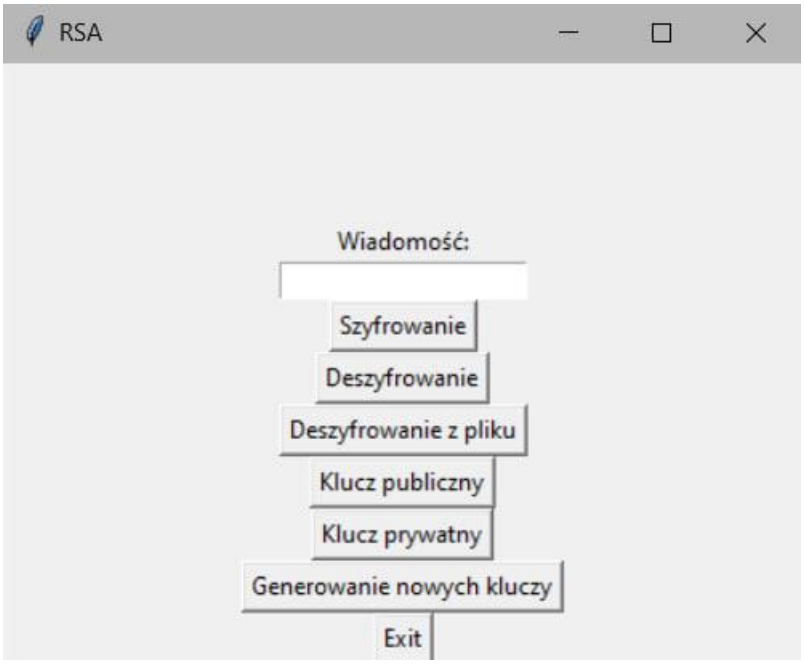
file.write(key.decode()) - zapisuje zawartość klucza do pliku jako ciąg znaków.

key = file.read() - odczytuje zawartość pliku i zapisuje ją do zmiennej "key".

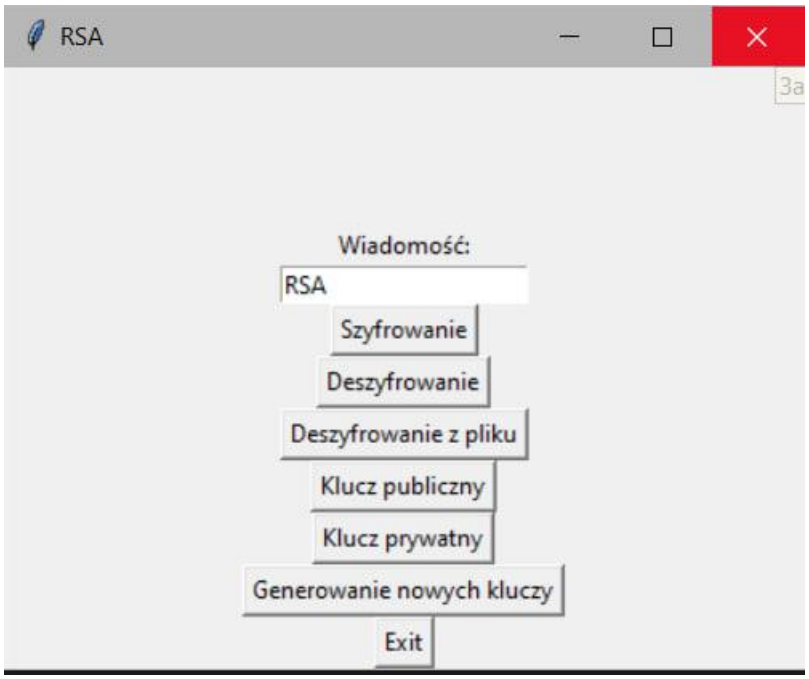
public_key = read_to_file('public_key.pem') – odczytuje plik "public_key.pem" i zapisuje go w zmiennej "public_key".*private_key = read_to_file('private_key.pem')* - odczytuje plik "privat_key.pem" i zapisuje go w zmiennej "privat_key".

private_key, public_key = read_to_file('private_key.pem'), read_to_file('public_key.pem') - odczytuje zawartość plików i zapisuje je do zmiennych.

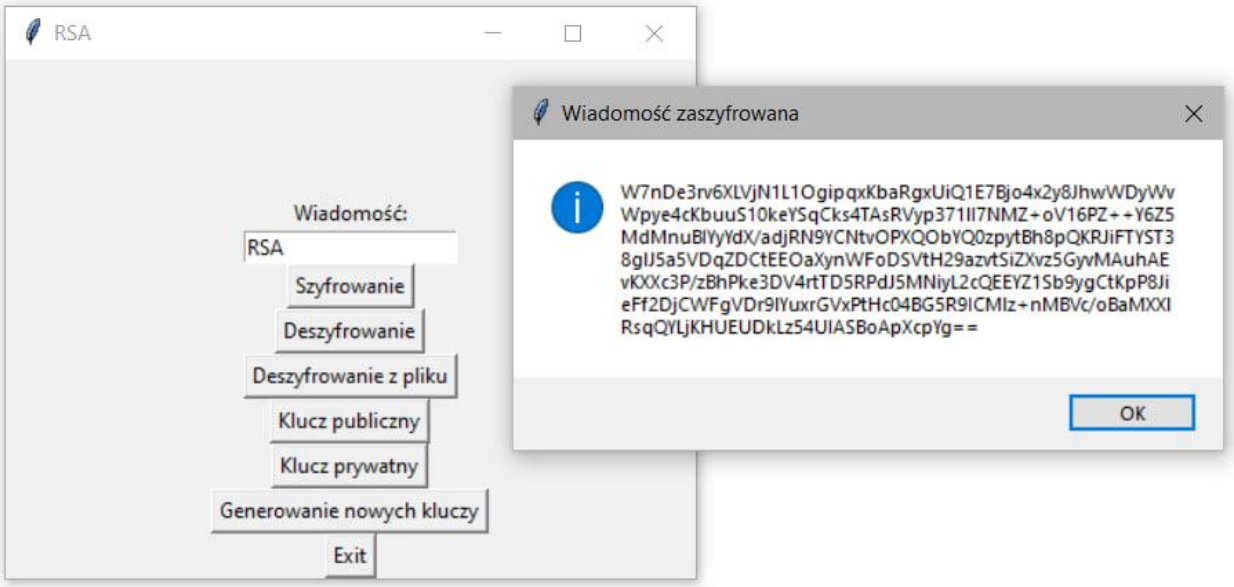
Po uruchomieniu aplikacji można przeglądać menu.



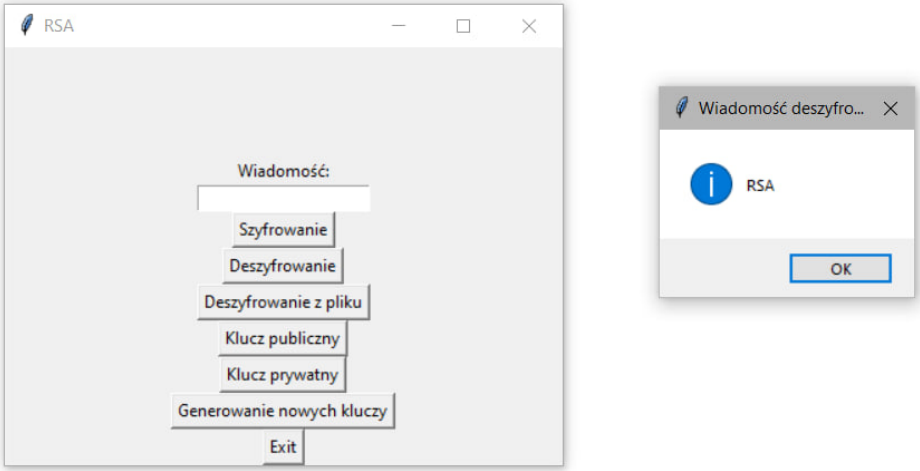
Następnie wprowadź wiadomość, którą chcesz zaszyfrować. Następnie naciśnij przycisk "Szyfrować".

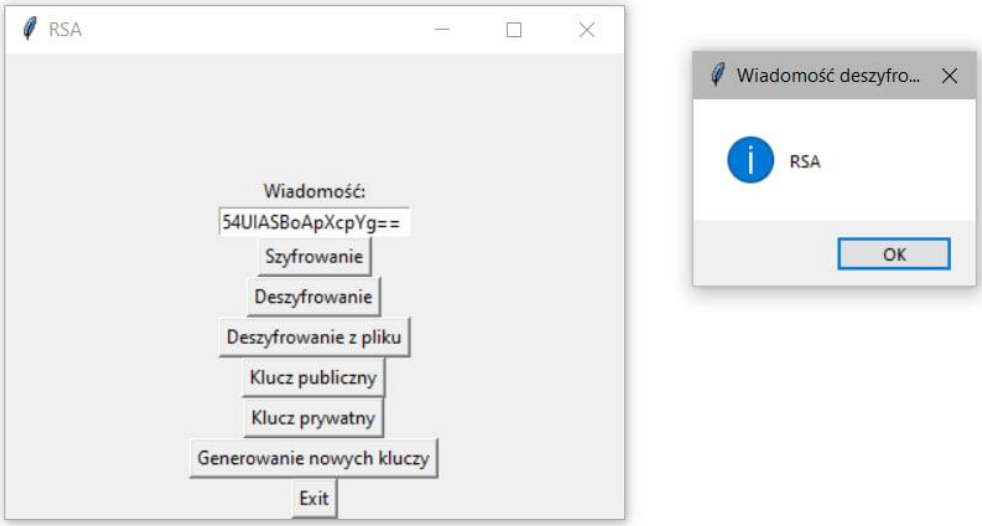


Na ekranie otrzymujemy zaszyfrowaną wiadomość, którą możemy wyświetlić w pliku "crypto.txt".

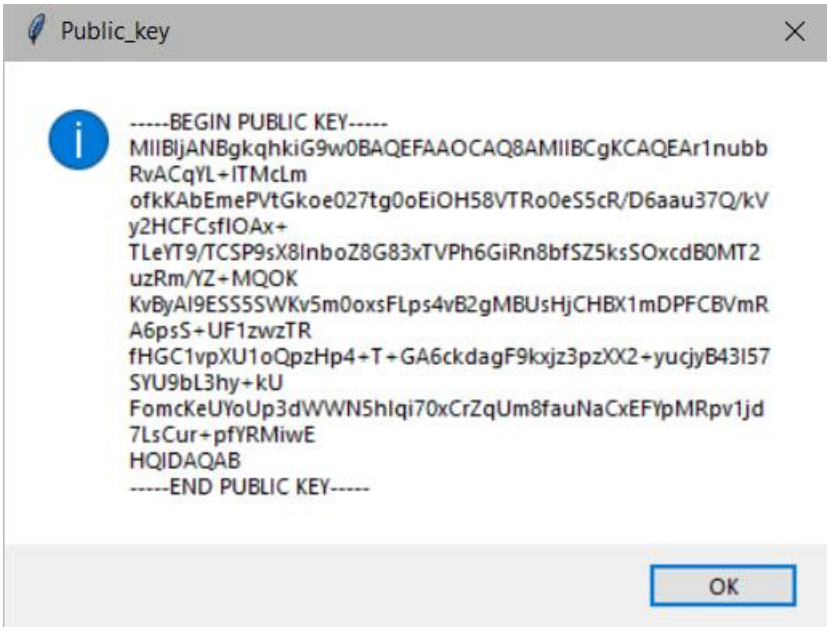


Kliknij „Deszyfrowanie z pliku”, aby odszyfrować zaszyfrowaną wiadomość z pliku "crypto.txt". Lub skopiuj zaszyfrowaną wiadomość z tego plika i wklej ją do pola



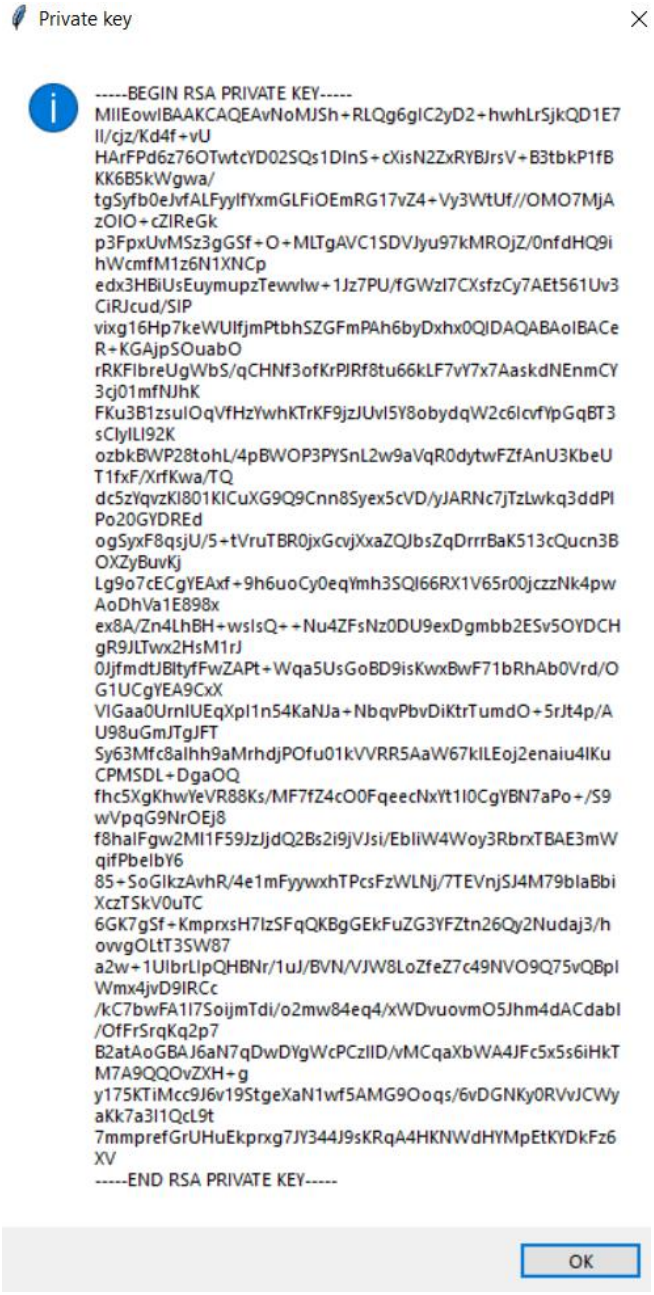


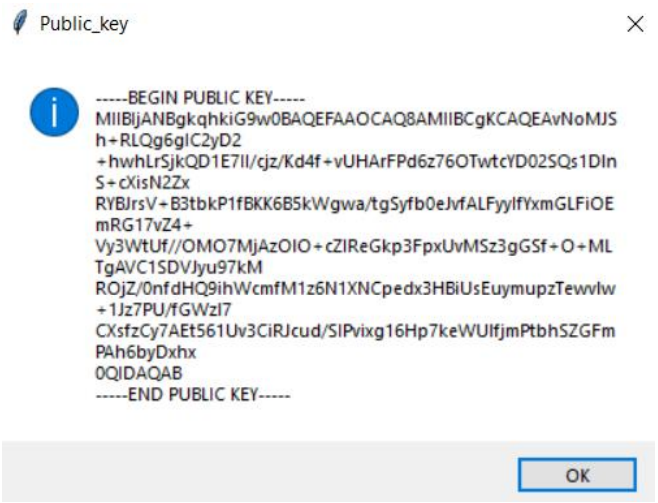
Naciśnij przycisk "Klucz publiczny".



Naciśnij przycisk "Klucz prywatny".

Aby wygenerować nowe klucze (publiczne i prywatne), naciśnij przycisk "'Generowanie nowych kluczy'".





Wnioski:

W tym projekcie stworzyliśmy system szyfrowania wykorzystujący algorytm RSA i szyfrowanie kluczem publicznym. Projekt ten zapewnia bezpieczną transmisję danych poprzez zapewnienie, że dane zaszyfrowane przy użyciu klucza publicznego mogą być odblokowane tylko przy użyciu odpowiedniego klucza prywatnego.

Generując unikalną parę kluczy, w której klucz prywatny jest utrzymywany w tajemnicy, a klucz publiczny może być dystrybuowany w bezpieczny sposób, możemy zapewnić poufność i integralność przesyłanych danych.

Ten projekt zapewnia prosty i wydajny sposób implementacji szyfrowania kluczem publicznym przy użyciu algorytmu RSA. Może on być wykorzystywany do ochrony poufnych informacji i zapewnienia bezpiecznej transmisji danych w różnych aplikacjach i systemach, w których bezpieczeństwo jest priorytetem.

Generując unikalną parę kluczy, pozwala użytkownikom na bezpieczną wymianę danych, ufając jedynie odpowiedniemu kluczowi prywatnemu do odblokowania zaszyfrowanych danych. Po prawidłowym wdrożeniu i przechowywaniu, projekt ten zapewnia silny i skuteczny mechanizm szyfrowania w celu zapewnienia prywatności i bezpieczeństwa danych.