

# Dokumentacja projektu nr.15, Implementacja aplikacji do szyfrowania plików z wykorzystaniem kryptografii asymetrycznej

Autorzy:

*Hanna Banasiak 193078,*

*Aleksandra Bujny 193186,*

*Marcel Grużewski 193589,*

*Michał Pawłojć 193159,*

*Jakub Romanowski 193637*

# Spis treści

1. Wstęp
  - 1.1. Szyfrowanie plików
  - 1.2. Algorytmy szyfrujące
2. Użytkowanie aplikacji
  - 2.1. Uruchomienie i podstawy
  - 2.2. Szyfrowanie
  - 2.3. Deszyfrowanie
  - 2.4. Dodawanie użytkownika
3. Aplikacja
  - 3.1. Działanie aplikacji
    - 3.1.1. Szyfrowanie
    - 3.1.2. Deszyfrowanie
    - 3.1.3. Dodanie użytkownika
  - 3.2. Moduły i biblioteki
  - 3.3. Algorytmy kryptograficzne
    - 3.3.1. Szyfrowanie symetryczne
    - 3.3.2. Szyfrowanie asymetryczne
  - 3.4. Struktura plików
  - 3.5. Funkcje
    - 3.5.1. main.py
    - 3.5.2. crypto\_sym.py
    - 3.5.3. crypto\_asym.py
4. Źródła

# 1. Wstęp

## 1.1. Szyfrowanie plików

Szyfrowanie plików ma na celu zabezpieczenie i ochronę przed ewentualnym naruszeniem danych. Jedynie nadawca i odbiorca danych mogą odczytać przesyłane dane, zaś osoby nieautoryzowane nie będą w stanie dostać się do treści, a jedynie do szyfrogramu, którego nie można odczytać bez posiadania klucza deszyfrującego. Zapobiega to naruszeniu integralności danych czy szkodliwym działaniom, takim jak np. ataki przechwytyjące.

## 1.2. Algorytmy szyfrujące

Szyfrowanie to zamiana tekstu jawnego przy pomocy algorytmu kryptograficznego na dane w postaci zaszyfrowanej (szyfrogram). Algorytmy dzielimy na symetryczne i asymetryczne.

Algorytmy symetryczne, ze względu na stosunkowo niewielką liczbę obliczeń, działają szybciej i są bardziej wydajne. Korzystają z jednego klucza zarówno do szyfrowania, jak i odszyfrowywania danych, co może prowadzić do utraty bezpieczeństwa, gdy klucz stanie się dostępny publicznie. Algorytmy można podzielić na dwa typy: strumieniowe - gdy wiadomość przetwarzana jest po pojedynczych bitach z użyciem operacji logicznej XOR oraz blokowe - gdy przetwarzane są bloki bitów o identycznej długości.

Algorytmy asymetryczne wymagają osobnych kluczy do szyfrowania, jak i deszyfrowania, co czyni je bezpieczniejszymi. Przechwycenie klucza publicznego (szyfrującego) nie stanowi ryzyka, ponieważ nie daje on możliwości odszyfrowania zabezpieczonych wiadomości. Klucz deszyfrujący (prywatny), posiada natomiast tylko odbiorca wiadomości. Algorytmy asymetryczne działają jednak wolniej, ze względu na dłuższe i bardziej skomplikowane obliczenia.

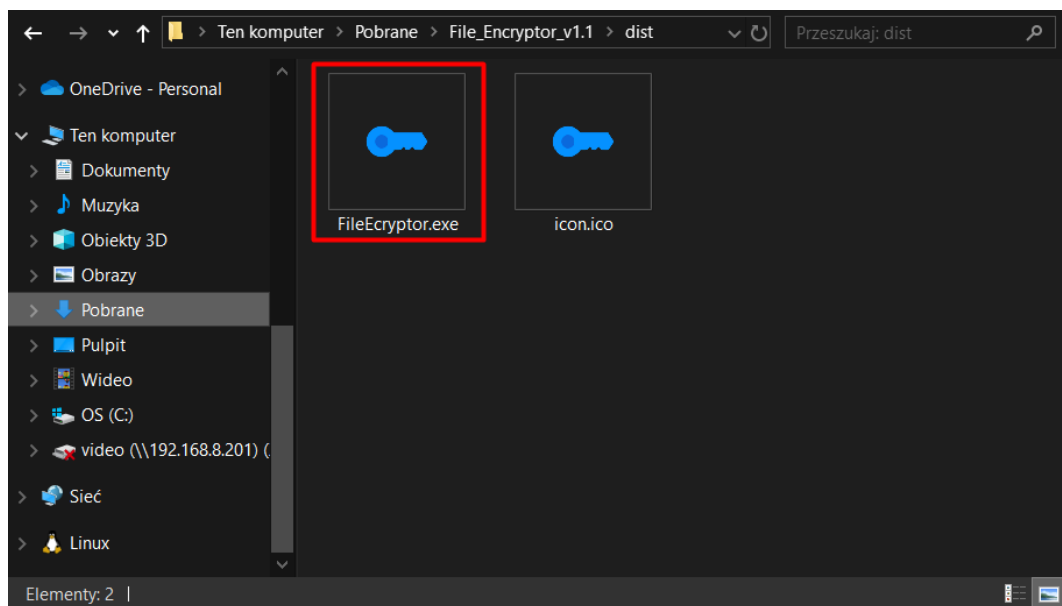
Można połączyć zalety obu algorytmów wykorzystując podejście hybrydowe. Pliki zaszyfrowywane są symetrycznie, co pozwala zachować wydajność. Przekazując klucz, musimy jednak zachować odpowiednie środki zabezpieczeń. Z tego względu klucz symetryczny szyfrowany jest dodatkowo algorytmem asymetrycznym i dopiero wtedy przekazywany użytkownikowi wraz z kluczem prywatnym.

## 2. Użytkowanie aplikacji

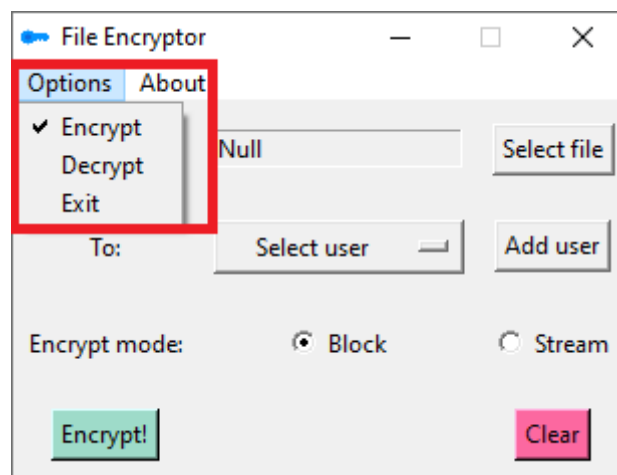
Celem projektu jest dostarczenie użytkownikowi aplikacji pozwalającej na szyfrowanie i deszyfrowanie wybranego przez niego pliku przy użyciu jednej z dwóch metod, aby następnie wysłać go do innego użytkownika.

### 2.1. Uruchomienie i podstawy

Aplikacja dostarczana jest użytkownikowi za pomocą pliku wykonywalnego (z rozszerzeniem .exe), który należy pobrać, a następnie kliknąć dwukrotnie na jego ikonę.

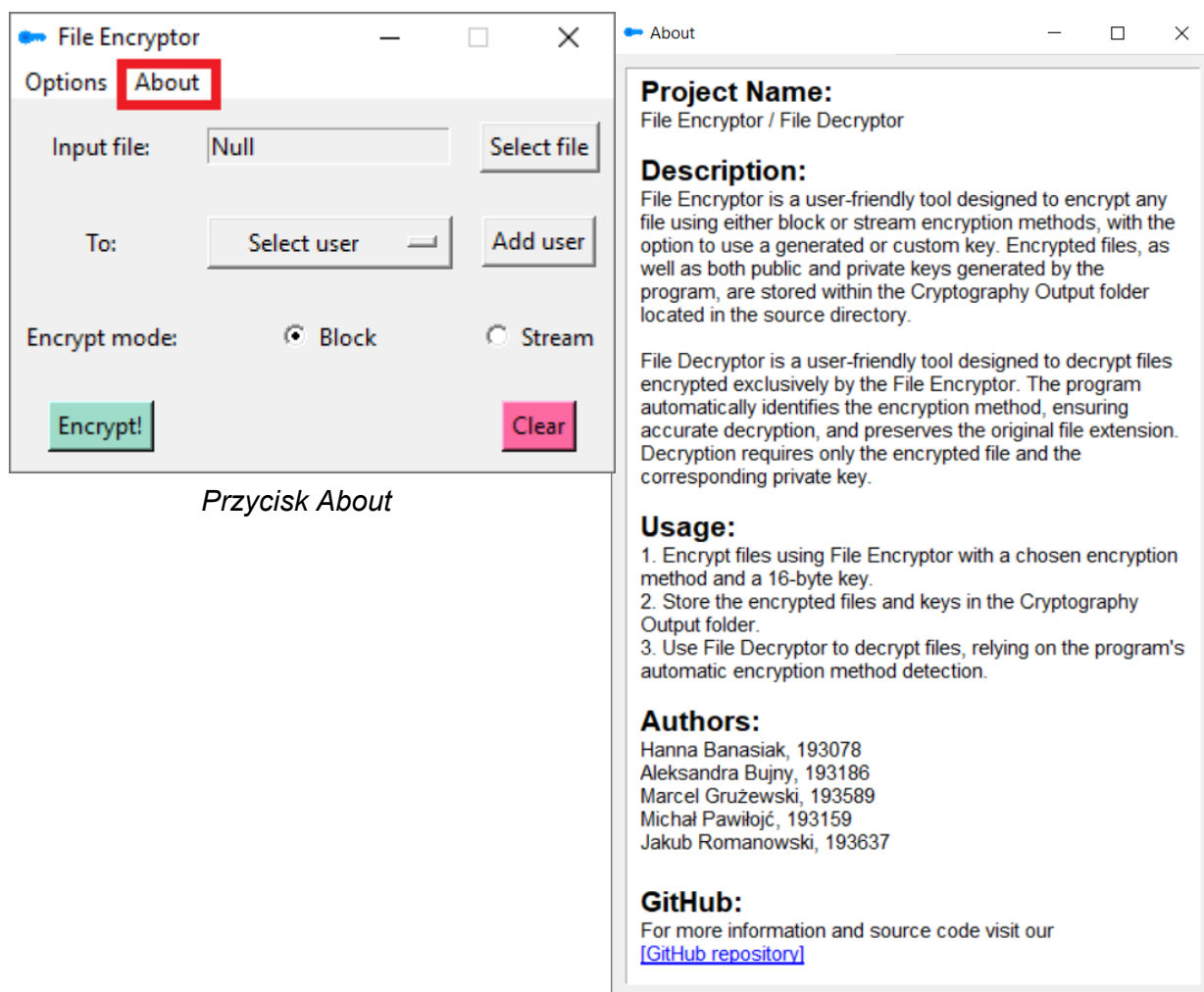


Program po uruchomieniu automatycznie otwiera opcję szyfrowania. Można przejść do deszyfrowania klikając w opcję *Decrypt* w opcjach w pasku menu w górnym lewym rogu aplikacji (*Options > Decrypt*). Można też wrócić do opcji szyfrowania wybierając *Encrypt*. (*Options > Encrypt*). Można również zamknąć aplikację wybierając opcję *Exit* (*Options > Exit*).



Menu opcji

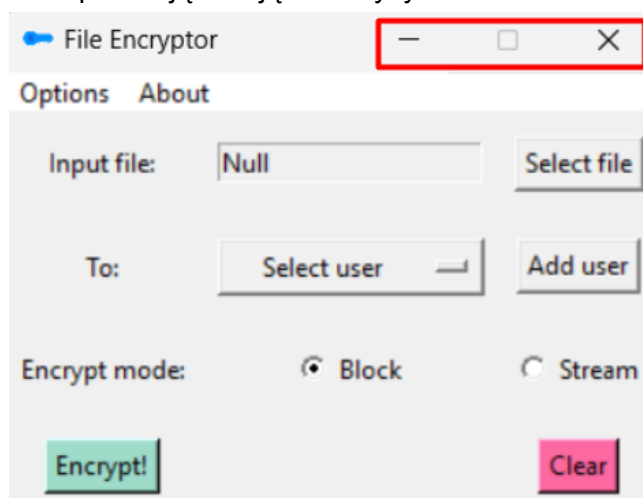
Po kliknięciu w przycisk *About* w pasku menu w górny lewy róg aplikacji pojawi się okienko zawierające informacje o aplikacji i autorach.



*Przycisk About*

*Sekcja About*

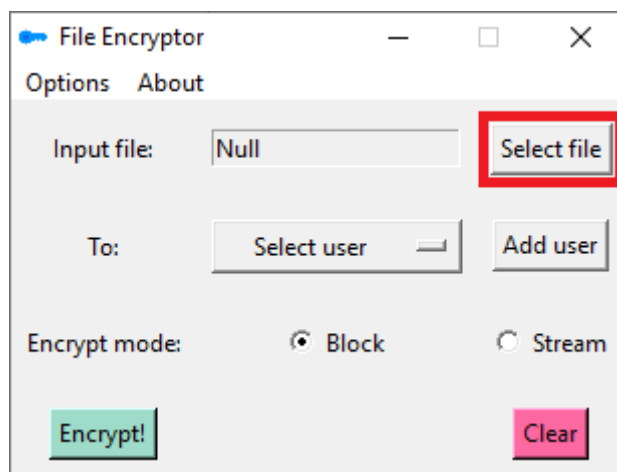
Aplikację można zminimalizować klikając w ikonkę poziomej kreski w prawym górnym rogu aplikacji lub opuścić ją klikając w krzyżyk.



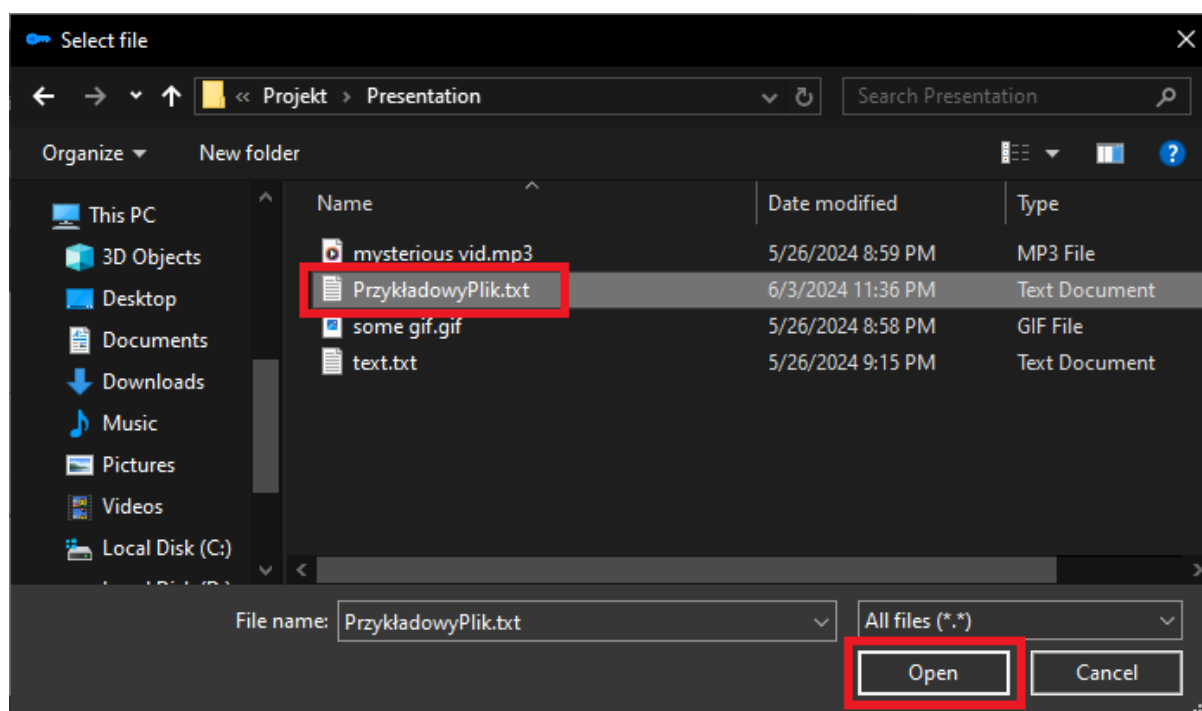
*Przyciski minimalizacji i wyjścia z aplikacji*

## 2.2. Szyfrowanie

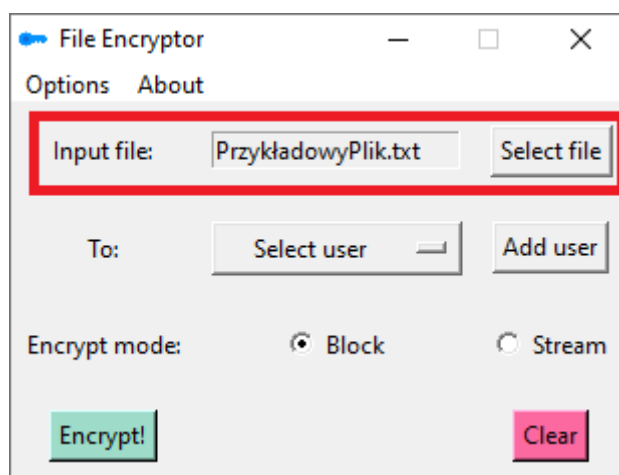
W celu zaszyfrowania pliku należy wybrać opcję szyfrowania (*Options > Encrypt*). Następnie w miejscu *Input file* należy wybrać plik do zaszyfrowania. Można to zrobić klikając w przycisk *Select file*, znajdując odpowiedni plik w Eksploratorze Plików i zatwierdzając wybór. Wybrany plik będzie widoczny w aplikacji.



*Przycisk wyboru pliku do zaszyfrowania*

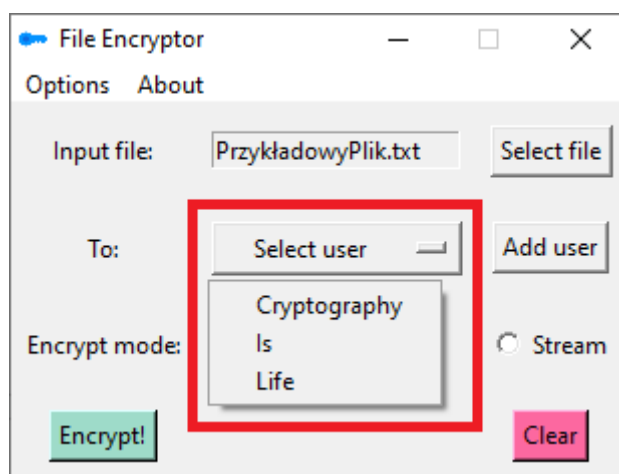


*Wybór pliku do zaszyfrowania*



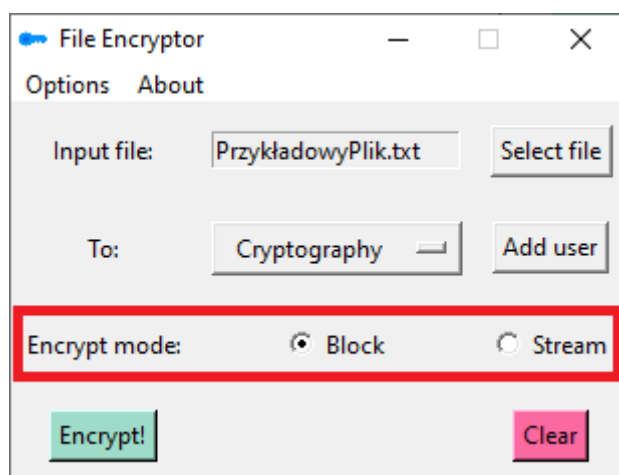
*Wybrany plik do zaszyfrowania*

Kolejnym krokiem jest wybranie w miejscu *To* odbiorcę pliku. Można go wybrać z listy istniejących użytkowników w miejscu *Select user* lub utworzyć nowego ([2.4.](#)).



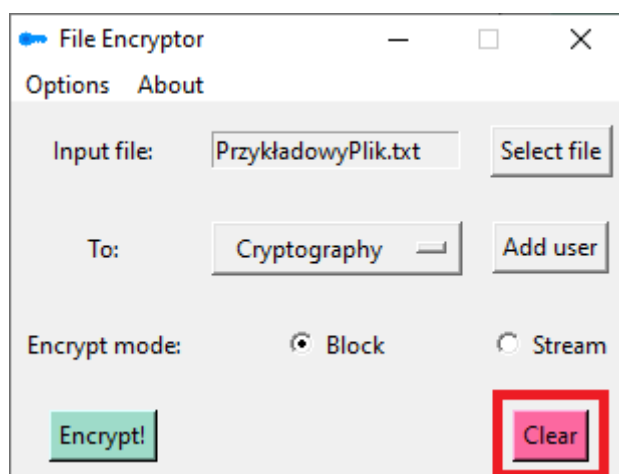
*Lista przykładowych użytkowników*

Ostatnim etapem jest wybranie sposobu szyfrowania. W miejscu *Encrypt mode* dostępne są dwie opcje: szyfrowanie blokowe (*Block*) i strumieniowe (*Stream*). Opcja jest domyślnie ustawiona na *Block*. Wybór będzie widoczny w aplikacji.

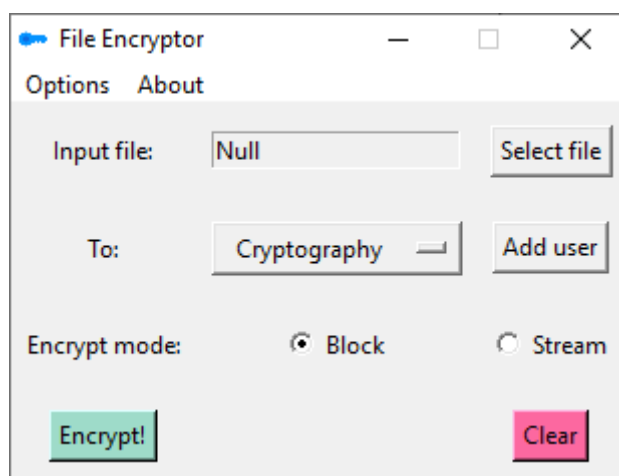


*Wybór sposobu szyfrowania*

W przypadku pomyłki można edytować wybrane opcje lub kliknąć przycisk *Clear* zaznaczony na różowo, który ustawi wszystkie opcje na domyślne.



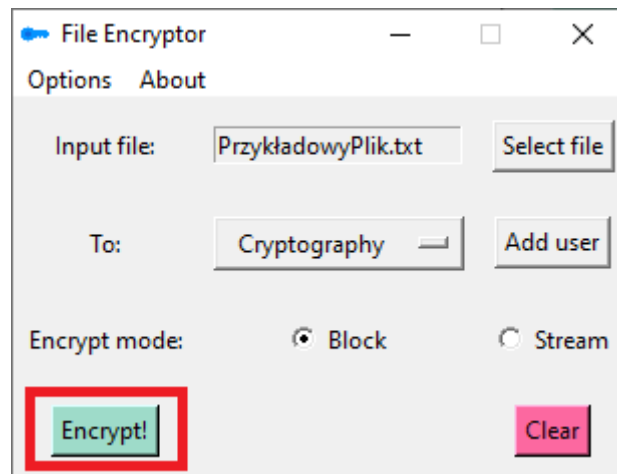
*Przycisk resetujący opcje*



*Okno aplikacji po użyciu przycisku Clear*



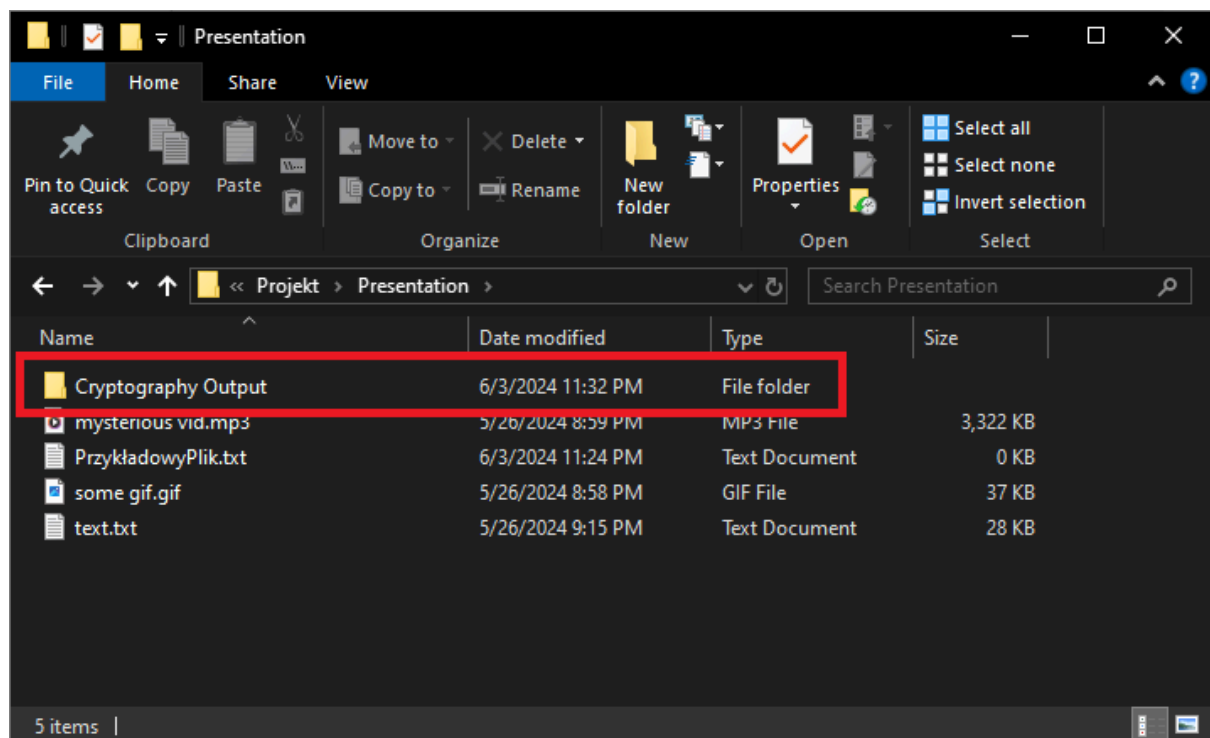
Aby zaszyfrować plik należy kliknąć opcję *Encrypt!* zaznaczoną na zielono. W ścieżce pliku szyfrowanego pojawi się folder *Cryptography Output*, w którym znajdziemy zaszyfrowany plik.



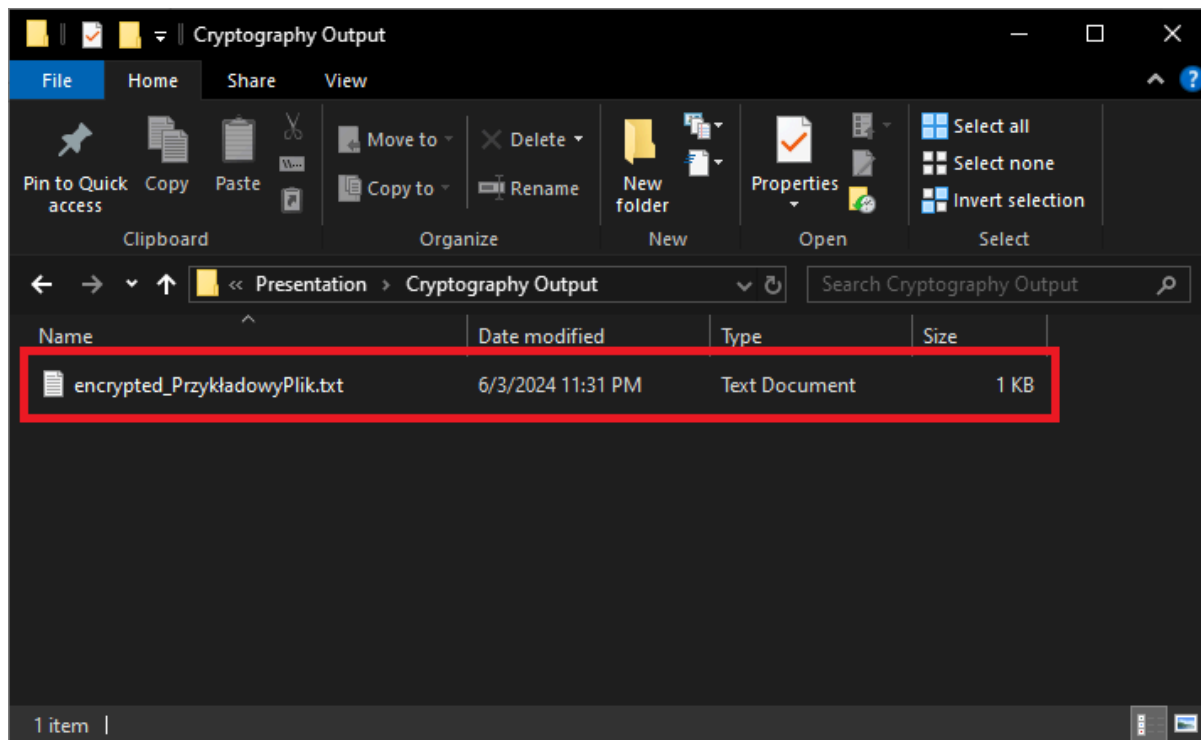
*Przycisk Encrypt!*



*Powiadomienie o sukcesie*



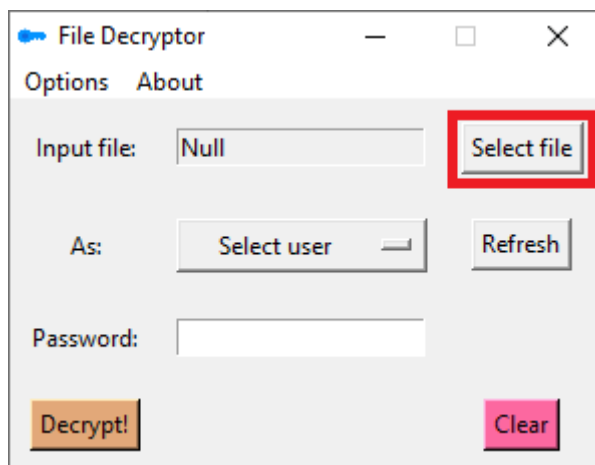
*Folder wyjściowy*



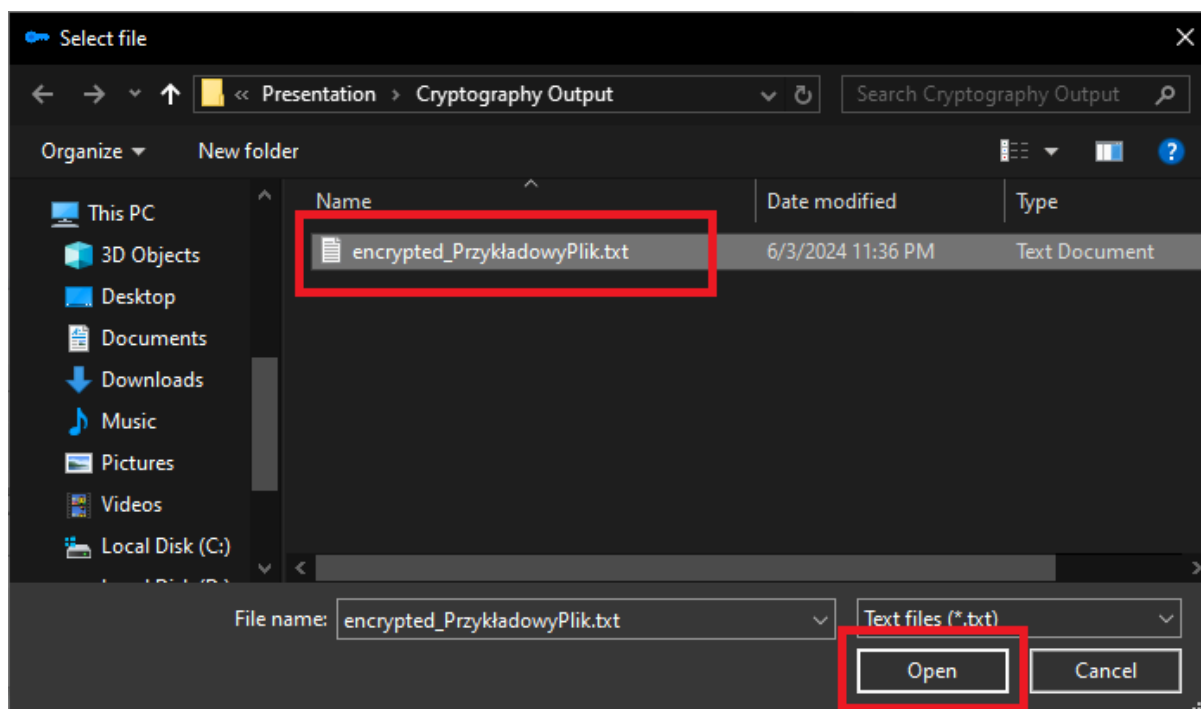
*Zaszyfrowany plik w folderze Cryptography Output*

## 2.3. Deszyfrowanie

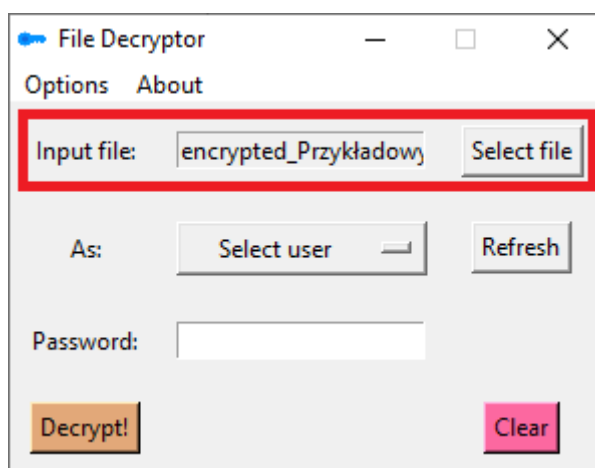
W celu odszyfrowania pliku należy wybrać opcję odszyfrowywania (*Options > Decrypt*). Następnie w miejscu *Input file* należy wybrać plik do odszyfrowania. Można to zrobić klikając w przycisk *Select file*, znajdując odpowiedni plik i zatwierdzając wybór. Wybrany plik będzie widoczny w aplikacji.



*Przycisk wyboru pliku do odszyfrowania*

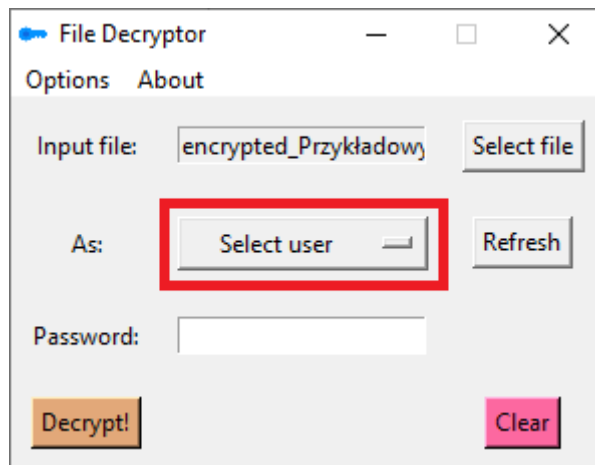


*Okno wyboru pliku do odszyfrowania*

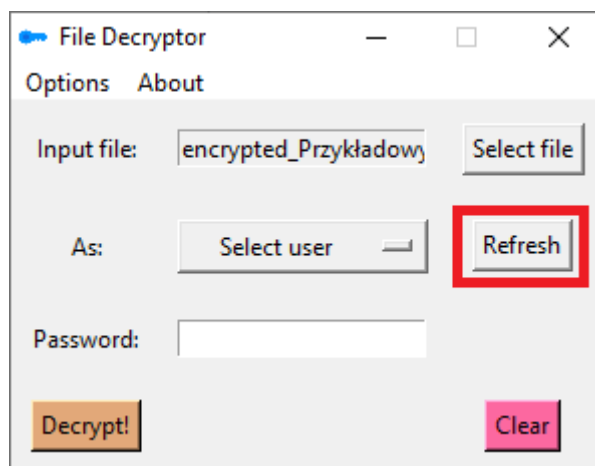


*Wybrany plik do odszyfrowania*

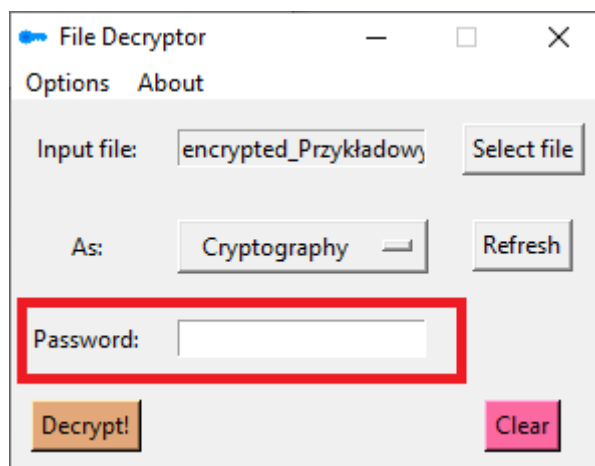
Kolejnym krokiem jest wybranie użytkownika w miejscu *As*. W miejscu *Select user* należy wybrać swoją nazwę użytkownika oraz podać hasło w miejscu *Password*. Jeśli użytkownik został dodany, a nie jest widoczny na liście dostępnych użytkowników można kliknąć przycisk *Refresh* w celu zaktualizowania listy.



*Przycisk wyboru użytkownika*



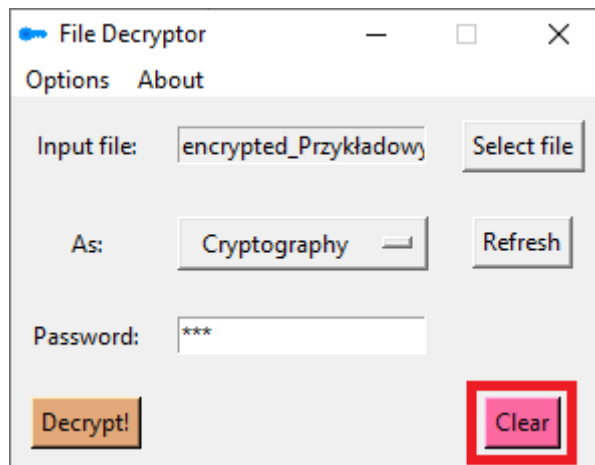
*Przycisk Refresh*



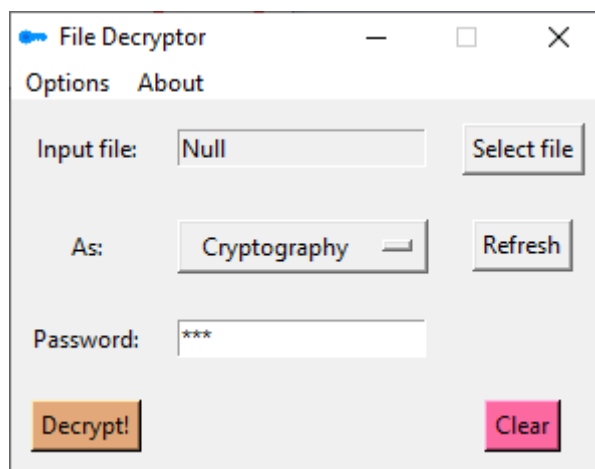
*Miejsce Password*

Przy odszyfrowywaniu pliku nie trzeba wybierać opcji szyfrowania. Aplikacja sama odszyfruje plik w odpowiedni sposób niezależnie od wybranej wcześniej opcji.

W przypadku pomyłki można wyczyścić pole z wybranym plikiem klikając w przycisk *Clear* zaznaczony na różowo.

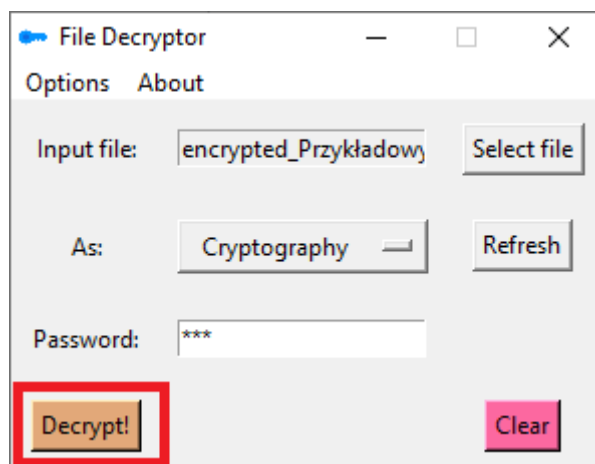


*Przycisk Clear*

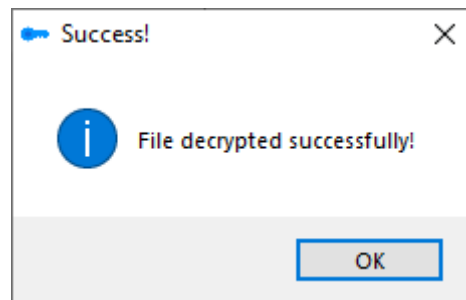


*Okno aplikacji po użyciu przycisku Clear*

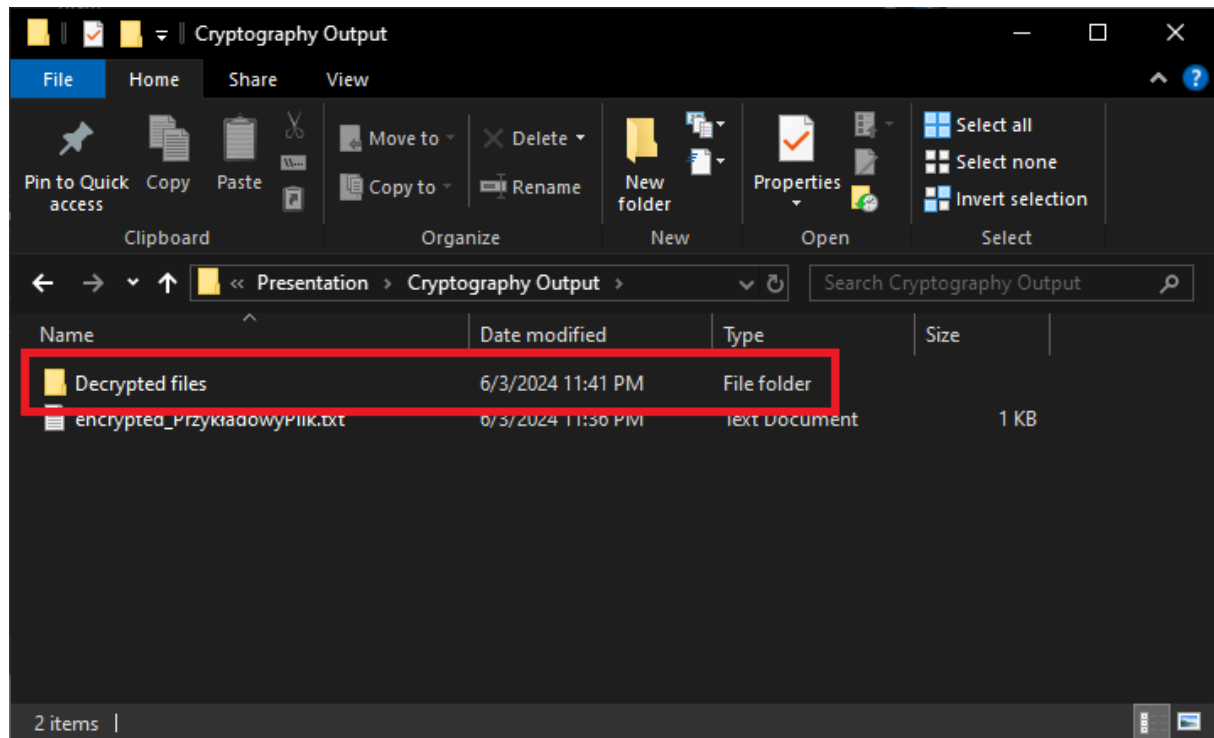
Po uzupełnieniu wszystkich pól należy kliknąć w przycisk *Decrypt!* oznaczony kolorem pomarańczowym. Jeżeli wszystkie pola są wypełnione poprawnie odszyfrowany plik pojawi się w folderze *Decrypted files*.



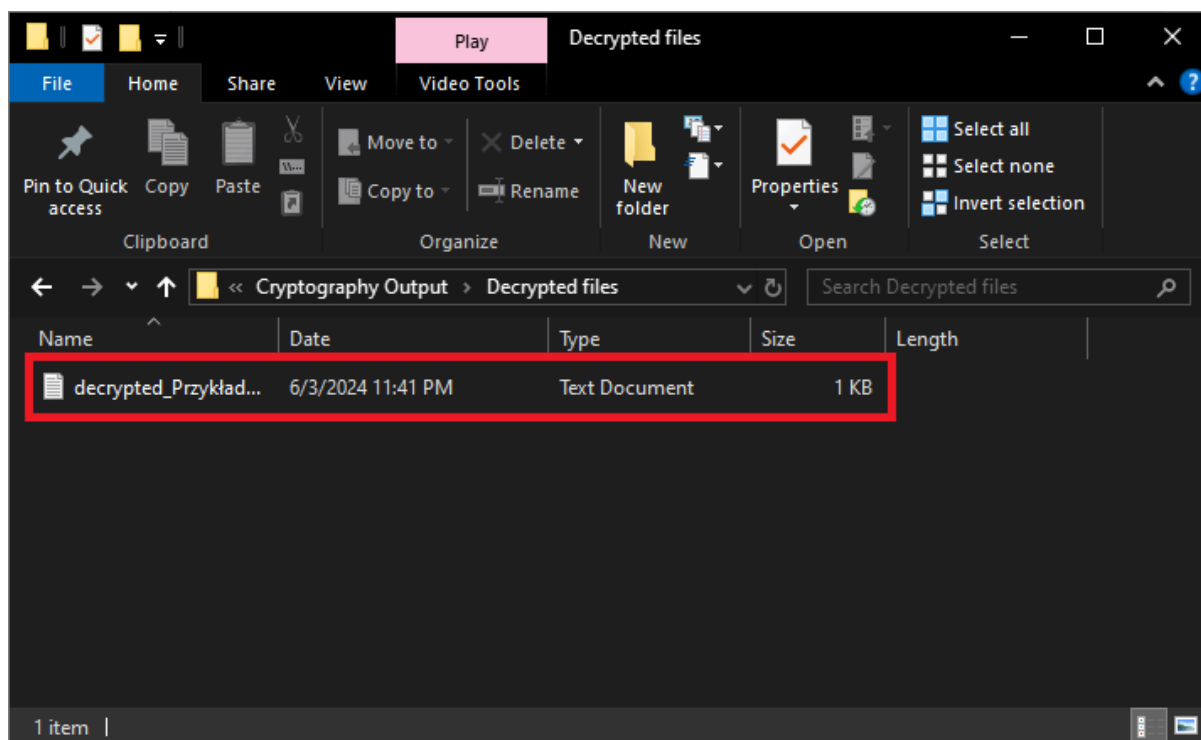
*Przycisk Decrypt!*



*Powiadomienie o sukcesie*



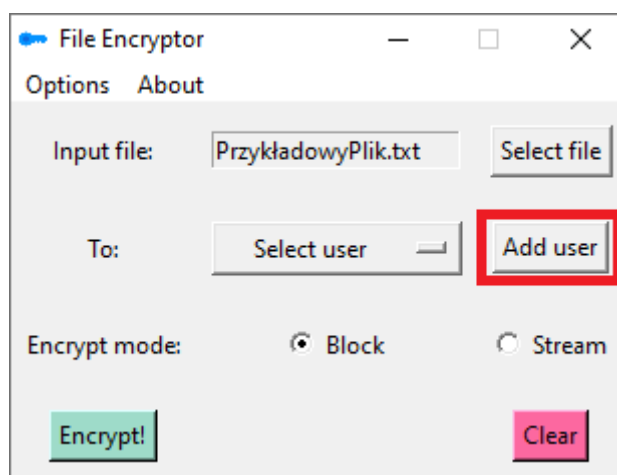
*Folder Decrypted files*



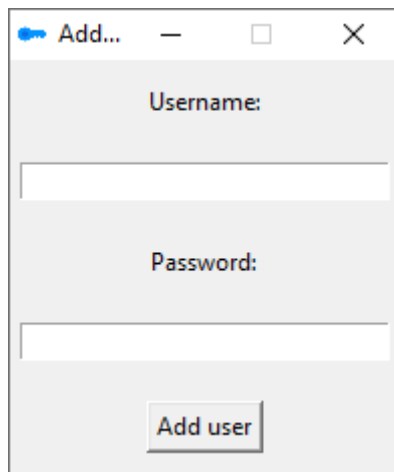
Odszyfrowany plik

## 2.4. Dodawanie użytkownika

W celu dodania nowego użytkownika należy wybrać opcję szyfrowania (*Options > Encrypt*). Następnie w miejscu *To* wybrać opcję *Add user*. Pojawi się nowe okno z polami *Username*, w którym należy wpisać nazwę nowego użytkownika, oraz *Password*, w którym należy wpisać hasło. Po uzupełnieniu pól należy kliknąć w przycisk *Add user*.



Przycisk wyboru użytkownika



Add...

Username:

Password:

Add user

*Okno dodawania użytkownika*



## 3. Aplikacja

Aplikacja wykorzystuje podejście hybrydowe, co pozwala na szybkie i bezpieczne operacje. Została zaimplementowana w języku Python, ze względu na prostotę implementacji oraz szeroką dostępność metod szyfrujących. Użyte zostało środowisko PyCharm.

### 3.1. Działanie aplikacji

#### 3.1.1. Szyfrowanie

1. Aplikacja oczekuje na podanie pliku do zaszyfrowania i trybu szyfrowania symetrycznego oraz wybrania odbiorcy pliku z rozwijanej listy. Jeśli odbiorcy pliku nie ma jeszcze na liście należy go stworzyć ([3.1.3](#)).
2. Plik zostaje zaszyfrowany symetrycznie z wykorzystaniem algorytmu blokowego CBC lub algorytmu strumieniowego CTR w zależności od wybranego trybu.
3. Aplikacja pobiera klucz publiczny odbiorcy pliku i za jego pomocą szyfruje asymetrycznie klucz symetryczny wraz z metadanymi (opisanymi w rozdziale [3.4](#)).
4. Zaszyfrowany symetrycznie plik wraz z zaszyfrowanym kluczem symetrycznym oraz metadanymi zapisywany jest na urządzeniu użytkownika, w tej samej ścieżce co oryginalny plik podany do zaszyfrowania

#### 3.1.2. Deszyfrowanie

1. Aplikacja wymaga wyboru pliku do odszyfrowania, wyboru z rozwijanej listy użytkownika, który chce odszyfrować plik oraz podania hasła.
2. Podane hasło, o ile jest poprawne, używane jest przez aplikację do deserializacji i odszyfrowania klucza prywatnego danego użytkownika, który następnie jest używany do odszyfrowania klucza symetrycznego. Dzięki temu, że w pliku znajdują się metadane, możliwe jest m.in. wykrycie przez aplikację trybu szyfrowania symetrycznego.
3. Po uzyskaniu stosownych danych niezbędnych do odszyfrowania zawartości oryginalnego pliku (tekstu jawnego) z metadanych, za pomocą klucza symetrycznego plik jest deszyfrowany, a następnie zapisywany na urządzeniu użytkownika w odpowiednim folderze (struktura katalogów opisana jest w rozdziale [3.4](#)).

#### 3.1.3. Dodawanie użytkownika

Jeśli odbiorcy pliku nie ma jeszcze na liście należy go stworzyć podając nazwę użytkownika oraz hasło, które później będzie konieczne do odszyfrowania serializowanego klucza prywatnego. Wygenerowana zostanie para kluczy asymetrycznych (publiczny i prywatny) dla nowego użytkownika i zostaną one umieszczone w katalogu (o nazwie identycznej jak użytkownika) na jego urządzeniu. Lista użytkowników na panelu szyfrowania sama się aktualizuje, natomiast przy panelu do odszyfrowywania dostępny jest przycisk odświeżania.

Jeśli użytkownik A chce coś wysłać do użytkownika B, a nie ma go na liście użytkowników, istotne jest, aby to użytkownik B się do niej dodał, ponieważ to on później będzie potrzebował hasła i pliku z kluczem prywatnym, a dane są zbyt wrażliwe, aby je przesyłać (zwiększa to ryzyko utraty prywatności). Kiedy konto użytkownika B zostanie już utworzone, powinien on przesłać plik z kluczem publicznym do użytkownika A. Po tym użytkownik A może przejść do zaszyfrowania pliku.

## 3.2. Moduły i biblioteki

Do stworzenia i poprawnego działania aplikacji wykorzystane zostały poniższe moduły i biblioteki.

|                                     |   |
|-------------------------------------|---|
| <a href="#"><u>tkinter</u></a>      | implementacja UI;<br>jego zaletą jest wieloplatformowość - działa zarówno na Windowsie, macOSie, jak i Linuxie                                  |
| <a href="#"><u>string</u></a>       | formatowanie ciągów znakowych   |
| <a href="#"><u>os</u></a>           | manipulacja plikami   |
| <a href="#"><u>secrets</u></a>      | generowanie kryptograficznie silnych liczb losowych, zamiast modułu <a href="#"><u>random</u></a> , który nie jest przeznaczony do kryptografii |
| <a href="#"><u>cryptography</u></a> | szyfrowanie i deszyfrowanie plików i kluczy   |
| <a href="#"><u>webbrowser</u></a>   | wyświetlanie repozytorium na GitHubie   |

Biblioteka [cryptography](#) podzielona jest na dwie części - wysokopoziomową, tzw. "recipes", która jest bezpieczna i łatwa w użyciu, a także nie wymaga konfiguracji oraz niskopoziomową - tzw. "cryptographic primitives" umieszczone w module *hazmat*. "Cryptographic primitives" to podstawowe, fundamentalne funkcjonalności, które zostały zaprojektowane do konkretnych, precyzyjnie określonych zadań. Charakteryzują się one wysoką niezawodnością i bezpieczeństwem. Projektowanie nowych prymitywów jest bardzo czasochłonne i podatne na błędy, nawet dla ekspertów w dziedzinie kryptografii. Dlatego w projekcie użyta została sprawdzona biblioteka, która została odpowiednio przetestowana. Wybrany został moduł *hazmat*, ponieważ zapewnia większą kontrolę nad używanymi prymitywami oraz pozwala m.in. na wybór trybu szyfrowania w szyfrowaniu symetrycznym.

Biblioteka [cryptography](#) nie posiada wad, które posiadają inne biblioteki kryptograficzne w Pythonie (np. M2Crypto, PyCrypto czy PyOpenSSL), takich jak:

- ★ niezwykle podatne na błędy API, brak API wysokopoziomowych i niepewne ustawienia domyślne
- ★ stosowanie słabych implementacji algorytmów - implementacji ze znanymi atakami typu side-channel
- ★ brak dalszego utrzymywania biblioteki
- ★ brak obsługi PyPy oraz Pythona 3

### 3.3. Algorytmy i funkcje kryptograficzne

Poniżej omówione zostaną funkcjonalności, klasy i algorytmy dostarczane przez bibliotekę *cryptography* użyte w projekcie.

#### 3.3.1. SZYFROWANIE SYMETRYCZNE:

**Cipher** - obiekt, który łączy algorytm szyfrujący (tutaj: **AES**) z trybem szyfrowania - **CBC** (blokowym) lub **CTR** (strumieniowym). Standardowo używany jest domyślny backend dostarczany przez bibliotekę (konkretna implementacja kryptograficzna), jednak możliwe jest także wprowadzenie innego backendu jako trzeciego argumentu.

Tryb szyfrowania **CBC** przyjmuje wektor inicjalizujący jako parametr. Musi on się składać z losowych bajtów (dokładnie tylu, ile rozmiar bloku) i być generowany ponownie przy każdym szyfrowaniu, nie ma jednak potrzeby, aby był tajny.

Tryb **CTR** również przyjmuje jako parametr "nonce", czyli jednorazowo używaną liczbę w postaci bajtowej, która nie musi być niejawna, natomiast istotne jest, aby była unikalna i nieużywana kilkakrotnie z tym samym kluczem.

**AES** (Advanced Encryption Standard) to symetryczny szyfr blokowy oparty na algorytmie Rijndaela i wykorzystujący operacje macierzowe. Rozmiar bloku określony jest na 128 bitów, natomiast rozmiar klucza - 128, 192 lub 256 bitów. Klucz ten musi być trzymany w tajemnicy i jest podawany jako argument.

Dostępne są metody takie jak **encryptor()** (dokonuje szyfrowania) oraz **decryptor()** (umożliwia deszyfrowanie), a wywołanie ich na obiekcie **Cipher** spowoduje, że wynik operacji będzie zgodny z interfejsem **CipherContext**, który przechowuje informacje o bieżącej operacji i pozwala na przetwarzanie danych w sposób ciągły. Wywołanie metody **update()** z danymi pozwala na aktualizację stanu operacji, a następnie **finalize()** - zakończenie operacji i uzyskanie danych.

Ponieważ tryb szyfrowania blokowy przyjmuje równe co do wielkości bloki danych, niekiedy konieczne jest wyrównanie bloku do żądanej długości - tzw. "padding". W aplikacji użyta została popularna klasa **PCKS7** - umożliwia ona dopełnianie bloku danych N bajtami będącymi znakiem unicode odpowiadającym wartości N, tak aby miał taki sam rozmiar, jak pozostałe bloki, używane w trybie szyfrowania **CBC**. Jako argument podajemy rozmiar pojedynczego bloku w bitach.

Funkcje **padder()** i **unpadder()** pozwalają odpowiednio na dopełnienie i cofnięcie dopełnienia bloku. Wywołanie tych funkcji skutkuje zastosowaniem zmian do interfejsu **PaddingContext**, które można zaktualizować metodą **update()** oraz zakończyć metodą **finalize()**.

Jeśli chodzi o generowanie klucza, biblioteka nie udostępnia do tego żadnej metody. Wykorzystany został moduł *secrets* oraz funkcja **randbelow()**, która umożliwia kryptograficznie bezpieczne losowanie liczb typu int w określonym przedziale.

### 3.3.2. SZYFROWANIE ASYMETRYCZNE:

**RSA** - algorytm asymetryczny, jeden z obecnie najpopularniejszych, opiera się na faktoryzacji (rozkładzie na czynniki) dużych liczb złożonych

Biblioteka udostępnia funkcję **generate\_private\_key()**, która przyjmuje argumenty takie jak:

- **public\_exponent**, czyli wskaźnik właściwości matematycznej używanej do generowania klucza. O ile nie ma specyficznej potrzeby należy użyć liczby pierwszej **65537**. Jest to liczba Fermata (czyli liczba naturalna, którą można przedstawić w postaci  $2^{2^n} + 1$ ) w prostej reprezentacji binarnej, co zapewnia dobrą wydajność jak i bezpieczeństwo w operacjach kryptograficznych.
- **key\_size**, czyli długość klucza prywatnego wyrażona w bitach. Nie może ona być mniejsza niż 512, a wartości poniżej 1024 są uznawane za możliwe do złamania. W aplikacji użyty został rekomendowany rozmiar 2024 bity.
- **backend**, czyli implementacja funkcji kryptograficznych, używany jest domyślny backend dostarczany przez bibliotekę

Funkcja ta zwraca nam instancję wygenerowanego klucza prywatnego **RSAPrivateKey**.

Posiada on między innymi takie metody jak:

- **decrypt()** - odszyfrowuje dane, konieczne jest podanie zaszyfrowanego tekstu oraz dopełnienia ("padding" - będącego instancją **AsymmetricPadding** opisanego niżej).
- **public\_key()** - zwraca instancję **RSAPublicKey**
- **private\_bytes()** - pozwala na serializację klucza do postaci bajtowej, aby później móc go zapisać, przyjmuje następujące argumenty:
  - **encoding** czyli wartość z typu wyliczeniowego **Encoding**, w projekcie użyty został typ kodowania **PEM** oparty na strukturze Base64 służącej do kodowania ciągu bajtów za pomocą ciągu znaków i przypisującej 64 wybranym znakom wartości od 0 do 63
  - **format** czyli wartość z typu wyliczeniowego **PrivateFormat**, w projekcie został użyty format **PKCS8**, który jest bardziej nowoczesny i pozwala na lepsze szyfrowanie niż często używany **TraditionalOpenSSL**
  - **encryption\_algorithm** - typ szyfrowania serializacji, użyty został typ **BestAvailableEncryption**, czyli szyfrowanie przy użyciu najlepszego dostępnego algorytmu, może się on zmieniać w czasie, obecnie używany jest aes-256-cbc. Dzięki temu klucz prywatny jest dodatkowo zabezpieczony i nawet jeśli dostanie się w niepowołane ręce, to bez znajomości hasła niemożliwe będzie odszyfrowanie pliku

Zserializowany klucz w formacie **PEM** możemy później załadować z pliku i zdeserializować przy pomocy metody **load\_pem\_private\_key()**.

Klucz publiczny **RSAPublicKey** zawiera następujące metody:

- **encrypt()** - zaszyfrowuje dane, należy podać dane do zaszyfrowania (w postaci bajtowej) oraz padding

- **public\_bytes()** - pozwala na serializację klucza publicznego, działa analogicznie do funkcji **private\_bytes()**, nie przyjmuje jednak argumentu **encryption\_algorithm**. Jeśli chodzi o użyty format jest nim **SubjectPublicKeyInfo**, czyli typowy format klucza publicznego jako ciąg bitowy, jest on rekomendowany do używania

**AsymmetricPadding** dostarcza nam różne schematy wyrównywania, ponieważ RSA szyfruje blokowo. W projekcie użyta została funkcja generowania masek **MGF1**, czyli funkcja odpowiadająca funkcji hashującej (funkcji skrótu), jednak różniąca się od niej tym, że zwraca wyniki różnej długości. Przyjmuje jako argument algorytm hashujący, w projekcie użyty został algorytm **sha256**.

### 3.4. Struktura plików

Plik zaszyfrowany złożony jest z symetrycznie zaszyfrowanej treści pliku oryginalnego oraz przechowywanego wraz z nią asymetrycznie zaszyfrowanego klucza symetrycznego wraz z metadanymi o następującej strukturze:

- Iv,
- Tryb szyfrowania symetrycznego,
- Rozszerzenie oryginalnego pliku,

Zaszyfrowane pliki zapisywane są w rozszerzeniu *.txt* i przedrostkiem *encrypted\_* w podfolderze *Cryptography Output* tworzonym w lokalizacji oryginalnego pliku szyfrowanego. Pliki odszyfrowane zapisywane są w oryginalnym rozszerzeniu z przedrostkiem *decrypted\_* w folderze *Cryptography Output/Decrypted files*

Struktura szyfrowanych plików dla *PrzykładowyPlik.txt*:

```
PrzykładowyPlik.txt
Cryptography Output/
├─ encrypted_PrzykładowyPlik.txt
└─ Decrypted files/
    └─ decrypted_PrzykładowyPlik.txt
```

Struktura plików do obsługi użytkowników dla użytkownika *PrzykładowyUżytkownik*:

```
FileEncryptor.exe
users/
└─ PrzykładowyUżytkownik/
    ├─ key.pub
    └─ key.priv
```

## 3.5. Funkcje w kodzie

### 3.5.1. main.py

| Funkcja                                     | Przyjmowane argumenty   | Działanie   |
|---|---|---|
| <b><i>get_file</i></b>                      | 1. <i>entry</i><br>2. <i>input_file</i><br>3. <i>file_name</i><br>4. <i>filetypes</i>       | Otwiera okno wyboru pliku i przypisuje nazwę i ścieżkę wybranego pliku odpowiednim zmiennym |
| <b><i>save_file</i></b>                     | 1. <i>entry</i><br>2. <i>file</i><br>3. <i>filetypes</i>                                    | Zapisuje podany plik odpowiedniego typu na odpowiedniej ścieżce                             |
| <b><i>are_variables_set</i></b>             | 1. <i>variables</i>   | Kontroluje argumenty; zwraca informację o błędzie   |
| <b><i>reset_form</i></b>                    | 1. <i>arguments</i>   | Ustawia domyślne argumenty aplikacji  |
| <b><i>change_frame</i></b>                  | 1. <i>frame_to_forget</i><br>2. <i>frame_to_add</i><br>3. <i>app_mode</i><br>4. <i>root</i> | Zmienia okno aplikacji w zależności od wyboru opcji szyfrowania lub deszyfrowania           |
| <b><i>create_encryption_output_path</i></b> | 1. <i>input_file</i>  | Tworzy odpowiedni folder przy szyfrowaniu   |
| <b><i>create_decryption_output_file</i></b> | 1. <i>input_file</i><br>2. <i>extension</i>   | Tworzy odpowiedni plik wyjściowy przy deszyfrowaniu   |
| <b><i>create_file_path</i></b>              | 1. <i>input_file</i><br>2. <i>folder_path</i>   | Tworzy odpowiednią ścieżkę wyjściową  |
| <b><i>create_encryption_UI</i></b>          | 1. <i>frame</i><br>2. <i>input_file</i><br>3. <i>key_value</i>                              | Tworzy okno aplikacji "Encrypt"   |
| <b><i>create_decryption_UI</i></b>          | 1. <i>frame</i><br>2. <i>input_file</i><br>3. <i>key_file</i>                               | Tworzy okno aplikacji "Decrypt"   |
| <b><i>encrypt</i></b>                       | 1. <i>input_file</i><br>2. <i>key_value</i><br>3. <i>encrypt_mode</i>                       | Obsługuje szyfrowanie plików  |

|                                  |  |   |
|----------------------------------|--|---|
| <b><i>decrypt</i></b>            | 1. <i>input_file</i><br>2. <i>key_file</i>   | Obsługuje deszyfrowanie plików                        |
| <b><i>show_about_section</i></b> | 1. <i>section_open</i>   | Tworzy okno aplikacji "About"                         |
| <b><i>add_user</i></b>           | 1. <i>is_menu_alive</i><br>2. <i>menu</i><br>3. <i>selected_user</i><br>4. <i>users_list</i> | Dodanie nowego użytkownika                            |
| <b><i>create_user_files</i></b>  |  | Utworzenie katalogu z kluczami użytkowników           |
| <b><i>finalize</i></b>           |  | Finalizuje tworzenie użytkownika                      |
| <b><i>refresh_dropdown</i></b>   | 1. <i>menu</i><br>2. <i>selected_user</i>  | Aktualizacja listy użytkowników                       |
| <b><i>get_user_keys</i></b>      | 1. <i>user</i>   | Pobiera klucz prywatny i publiczny danego użytkownika |
| <b><i>load_users</i></b>         |  | Wczytuje utworzonych użytkowników                     |

### 3.5.2. crypto\_sym.py

| Funkcja                         | Przyjmowane argumenty   | Działanie  |
|---------------------------------|---|--|
| <b><i>generate_key</i></b>      | 1. <i>key_value</i><br>2. <i>key_length</i>                             | Generuje klucz o wybranej długości i przypisuje do odpowiedniej zmiennej               |
| <b><i>random_characters</i></b> | 1. <i>key_length</i>  | Zwraca ciąg losowych znaków  |
| <b><i>encrypt_sym</i></b>       | 1. <i>data</i><br>2. <i>key_value</i><br>3. <i>mode</i>                 | Szyfruje plik symetrycznie;<br>Zwraca informacje o błędach w przypadku niepowodzenia   |
| <b><i>decrypt_sym</i></b>       | 1. <i>ciphertext</i><br>2. <i>key</i><br>3. <i>iv</i><br>4. <i>mode</i> | Deszyfruje plik symetrycznie;<br>Zwraca informacje o błędach w przypadku niepowodzenia |

### 3.5.3. crypto\_asym.py

| Funkcja                         | Przyjmowane argumenty                         | Działanie                               |
|---------------------------------|---|---|
| <b><i>generate_key_pair</i></b> |   | Generuje parę kluczy                    |
| <b><i>encrypt_asym</i></b>      | 1. <i>text</i><br>2. <i>key_pub</i>           | Szyfruje klucz publiczny asymetrycznie  |
| <b><i>decrypt_asym</i></b>      | 1. <i>ciphertext</i><br>2. <i>key_priv</i>    | Deszyfruje klucz prywatny asymetrycznie |
| <b><i>save_private_key</i></b>  | 1. <i>private_key</i><br>2. <i>filename</i>   | Tworzy plik klucza prywatnego           |
| <b><i>save_public_key</i></b>   | 1. <i>public_key</i><br>2. <i>filename</i>    | Tworzy plik klucza publicznego          |
| <b><i>load_private_key</i></b>  | 1. <i>filename</i><br>2. <i>password=None</i> | Odczytuje klucz prywatny z pliku        |
| <b><i>load_public_key</i></b>   | 1. <i>filename</i>                            | Odczytuje klucz publiczny z pliku       |

## 4. Źródła

Przy tworzeniu dokumentacji wykorzystane zostały poniższe źródła:

- [medium.com](https://medium.com) - artykuł "Hybrid encryption in python"
- [zpe.gov.pl](https://zpe.gov.pl) - artykuł "Szyfry symetryczne i asymetryczne"
- [bezpieczny.blog](https://bezpieczny.blog)
- [pl.wikipedia.org](https://pl.wikipedia.org)
- [cyberwiedza.pl](https://cyberwiedza.pl) - artykuł "Hybrid Encryption"
- [kanga.exchange](https://kanga.exchange) - artykuł "Czym jest szyfrowanie asymetryczne"
- [docs.python.org](https://docs.python.org) - opis modułu *secrets*
- [cryptography.io](https://cryptography.io) - dokumentacja biblioteki *cryptography*