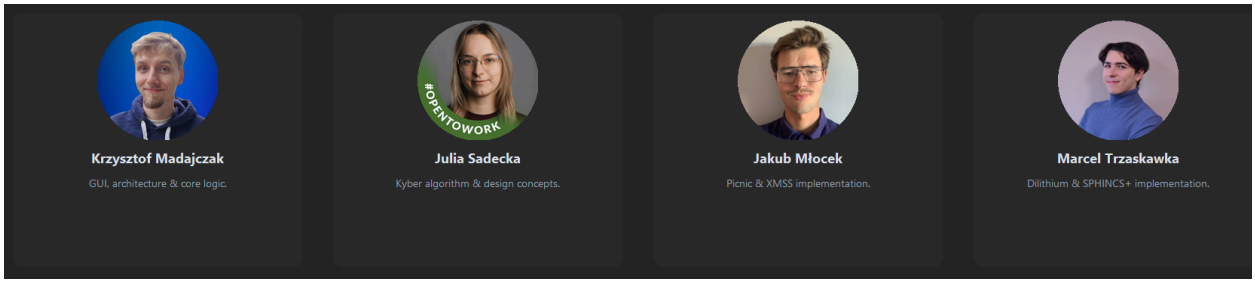
Aplikacja do podpisywania plików, weryfikacji podpisów, a także generowania kluczy. Wykorzystująca algorytmy: Kyber, Dilithium, Falcon, Cross, SPHINCS++.

styczeń 2026

Skład grupy



Harmonogram

1. Integracja wszystkich algorytmów z aplikacją - do 28.11
2. Dodanie funkcji zapisu klucza prywatnego na urządzeniu YubiKey - do 5.12
3. Przeprowadzenie testów funkcjonalnych i użytkowych - do 8.01
4. Opracowanie pełnej dokumentacji technicznej oraz przygotowanie końcowej prezentacji projektu - do 26.01

Podział prac

- Krzysztof Madajczak - Implementacja Gui, głównej struktury programu, wykrywanie oraz obsługa dysku zewnętrznego (pendrive)
- Julia Sadecka - Implementacja algorytmu szyfrującego Kyber, przygotowanie raportu diagramów UML, oraz prezentacji.
- Jakub Młocek - Implementacja algorytmu Falcon i Cross i przygotowanie benchmarków algorytmów szyfrujących i podpisu cyfrowego.
- Marcel Trzaskawka - Implementacja algorytmów podpisu cyfrowego; Obsługa headerów plików z celu automatycznego wykrycia algorytmów; Opis instalacji

Cel projektu

Celem projektu było stworzenie aplikacji umożliwiającej wygodne i bezpieczne korzystanie z algorytmów postkwantowych w codziennych operacjach kryptograficznych. Aplikacja pozwala na generowanie kluczy, podpisywanie i weryfikację dokumentów oraz szyfrowanie i deszyfrowanie plików, zapewniając jednocześnie intuicyjny interfejs użytkownika w którym możliwe jest także automatyczne wykrycie kluczy na nośnikach przenośnych. Dodatkowo, wbudowany moduł benchmarku umożliwia ocenę wydajności poszczególnych algorytmów, co pozwala porównać ich efektywność w praktyce.

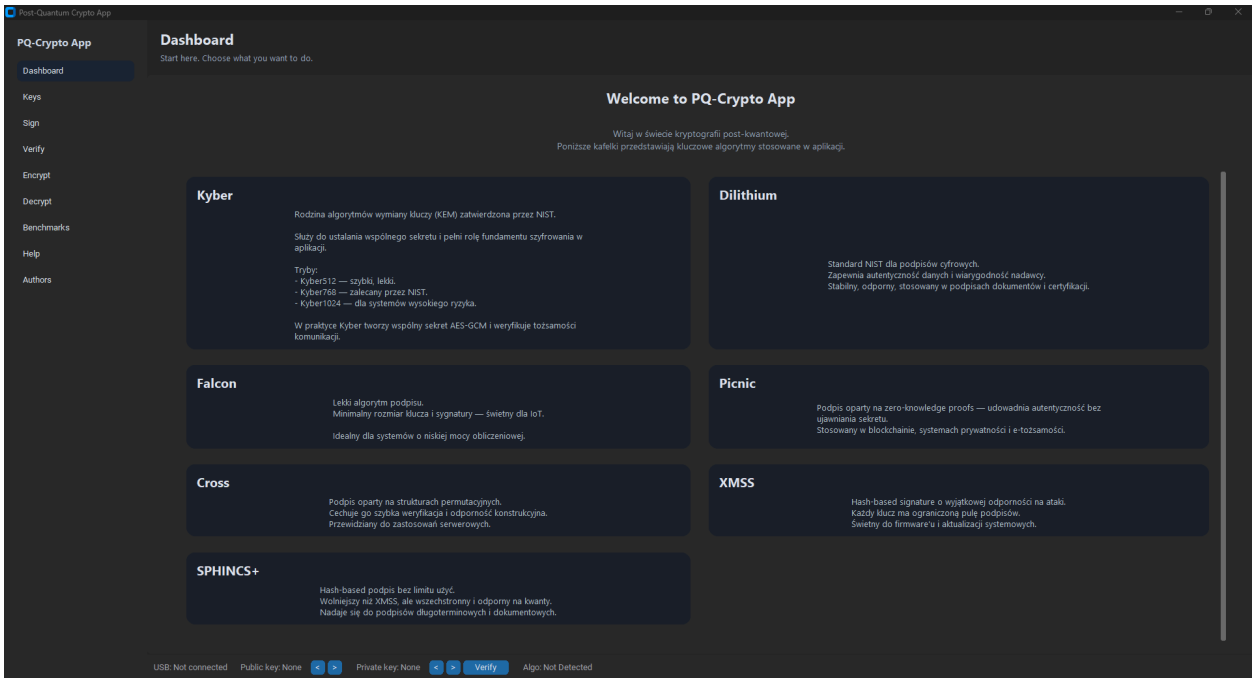
Szczegóły techniczne dotyczące instalacji wymaganych bibliotek zostały opisane w rozdziale „Instalacja”, a użyte algorytmy postkwantowe w rozdziale „Wybrane algorytmy”. Logika działania oraz pełna funkcjonalność aplikacji została przedstawiona w rozdziale „Funkcjonalność aplikacji”.

Funkcjonalność aplikacji

Nasza aplikacja ma kilka głównych funkcjonalności:

- generowanie kluczy przy użyciu algorytmów postkwantowych,
- odczyt kluczy z pamięci przenośnej (pendrive),
- podpis i weryfikacja plików,
- szyfrowanie i deszyfrowanie plików,
- benchmarki.

Cała aplikacja została napisana w języku programownia Python. Do GUI została wykorzystana biblioteka customtkinter [Dokumentacja], a do implementacji algorytmów postkwantowych użyliśmy biblioteki liboqs.



Obraz 1. Interfejs graficzny aplikacji (strona powitalna).

Diagram architektury systemu

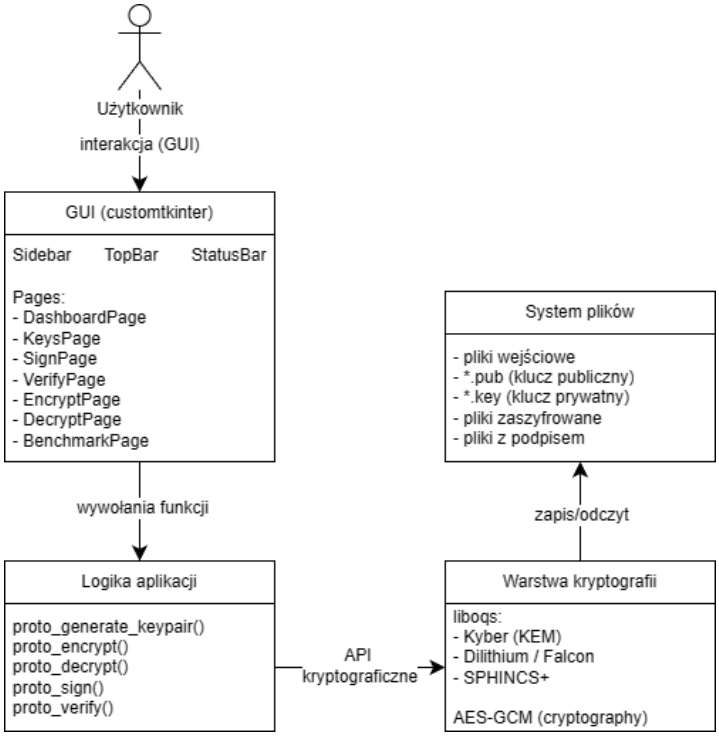
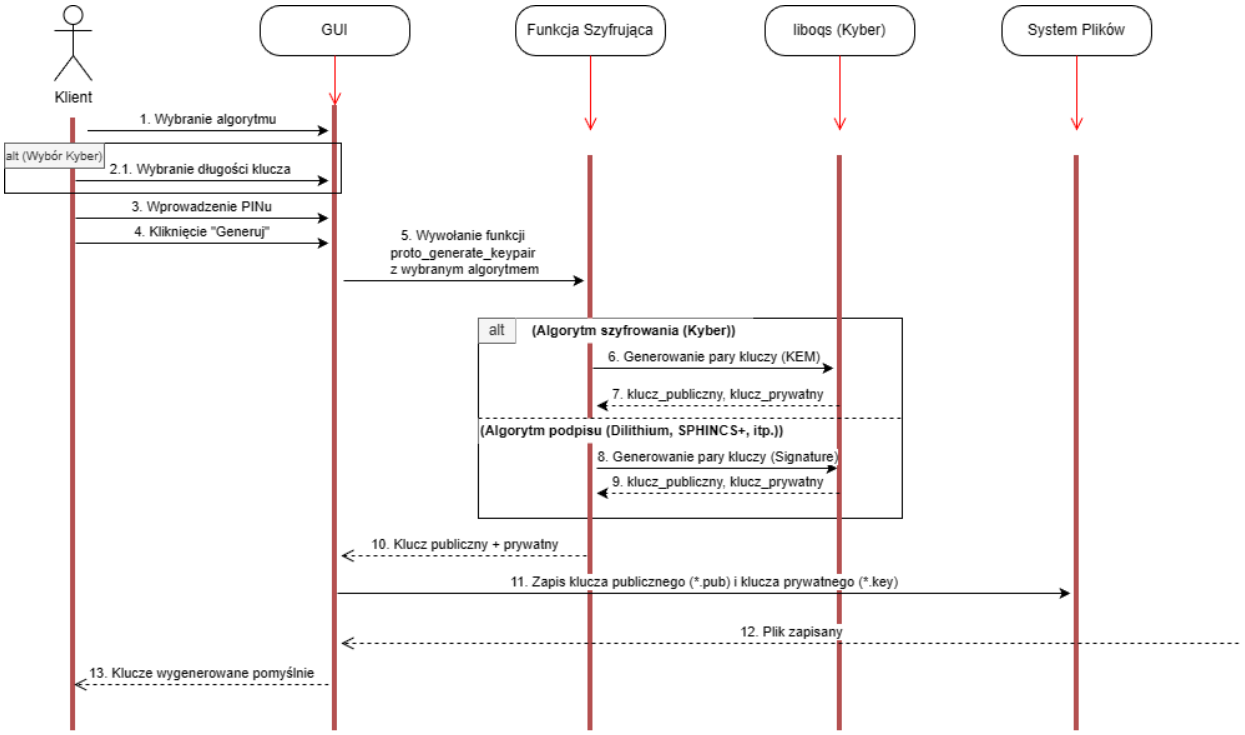


Diagram architektury systemu pokazuje w jaki sposób działa nasza aplikacja. Interfejs graficzny odpowiada za kontakt z użytkownikiem i obsługę jego działań, natomiast cała właściwa logika kryptograficzna została wydzielona do osobnego modułu, który korzysta z biblioteki liboqs. Operacje na plikach, takie jak zapis i odczyt kluczy czy danych, realizowane są przez system plików, który stanowi niezależny element architektury.

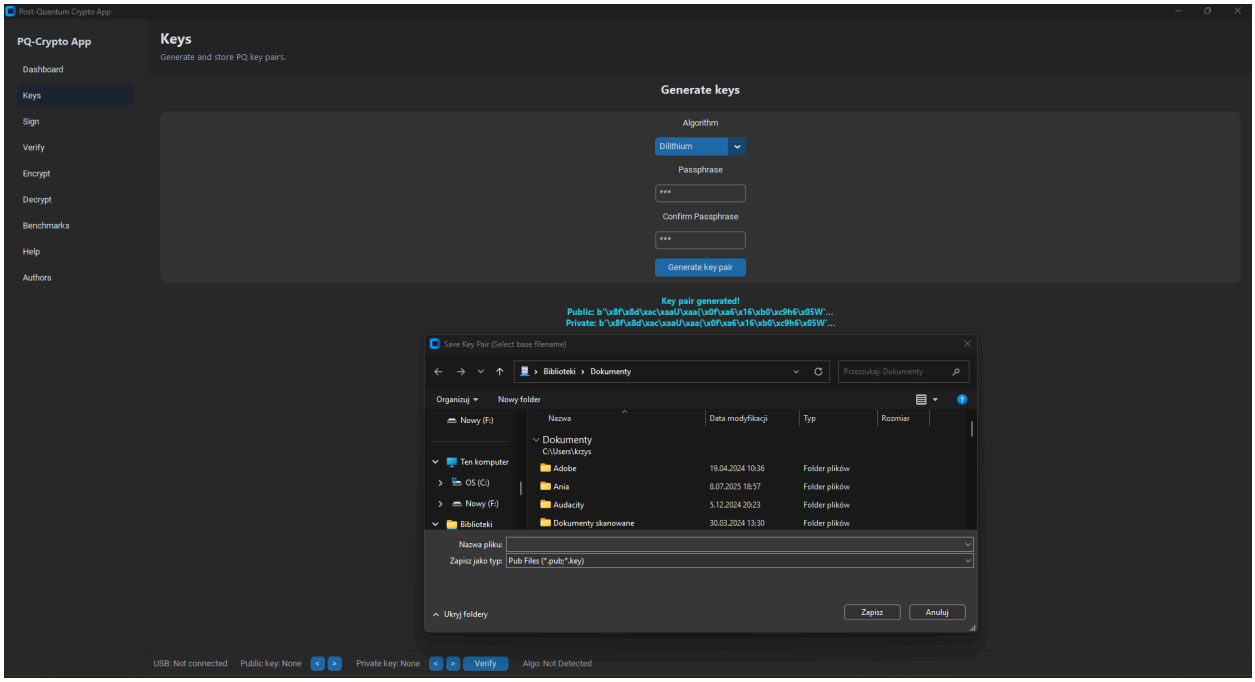
Generowanie Kluczy

Diagram sekwencji – generowanie kluczy

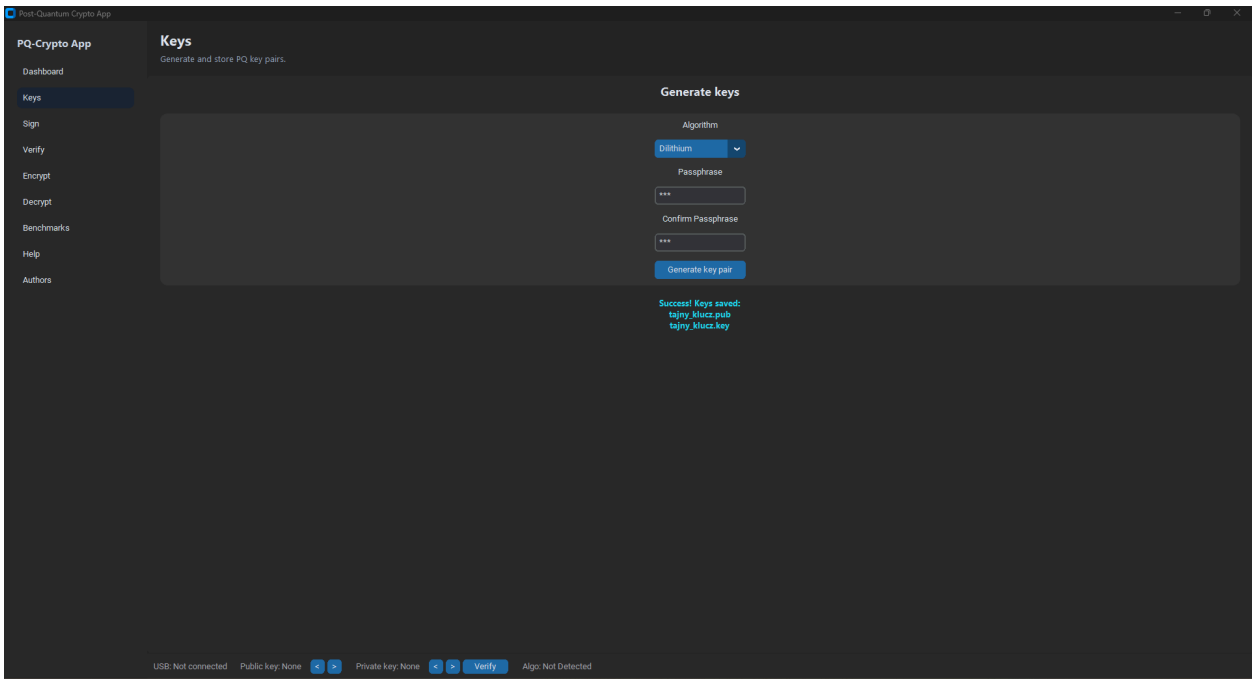


Generowanie kluczy zostało przedstawione na diagramie sekwencyjnym. W celu wygenerowania pary kluczy użytkownik musi wybrać algorytm kryptograficzny (domyślnie jest to Dilithium). W przypadku wyboru algorytmu Kyber użytkownik dodatkowo wybiera długość klucza (domyślnie 512). Następnie wprowadza on PIN i klika przycisk „Generuj”. Wywoływana jest funkcja *proto_generate_keypair(),* a użytkownikowi zwracana jest para kluczy: prywatny (.key) oraz publiczny (.pub).

W obu przypadkach na początku każdego pliku dodawany jest nagłówek w postaci nazwy algorytmu, oddzielonej spacją od właściwego klucza, dzięki czemu po wybraniu pliku system jest w stanie określić, jakiego algorytmu należy użyć. Przykład nagłówka wraz z kluczem przedstawiono na obrazie 6. Na obrazach 2 i 3 pokazano poprawny przebieg generowania kluczy od strony interfejsu graficznego, natomiast na obrazach 4 i 5 zaprezentowano sytuacje wyjątkowe, w których użytkownik nie wprowadził tego samego PIN-u jako potwierdzenia lub nie podał PIN-u w ogóle.



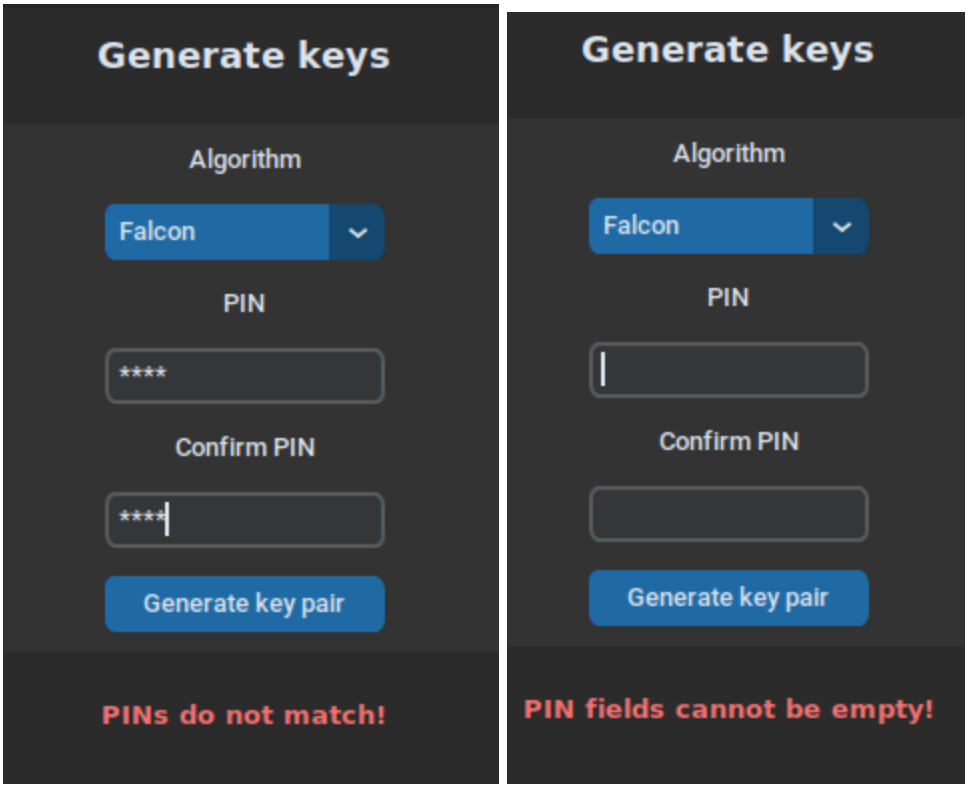
Obraz 2. Generowanie pary kluczy



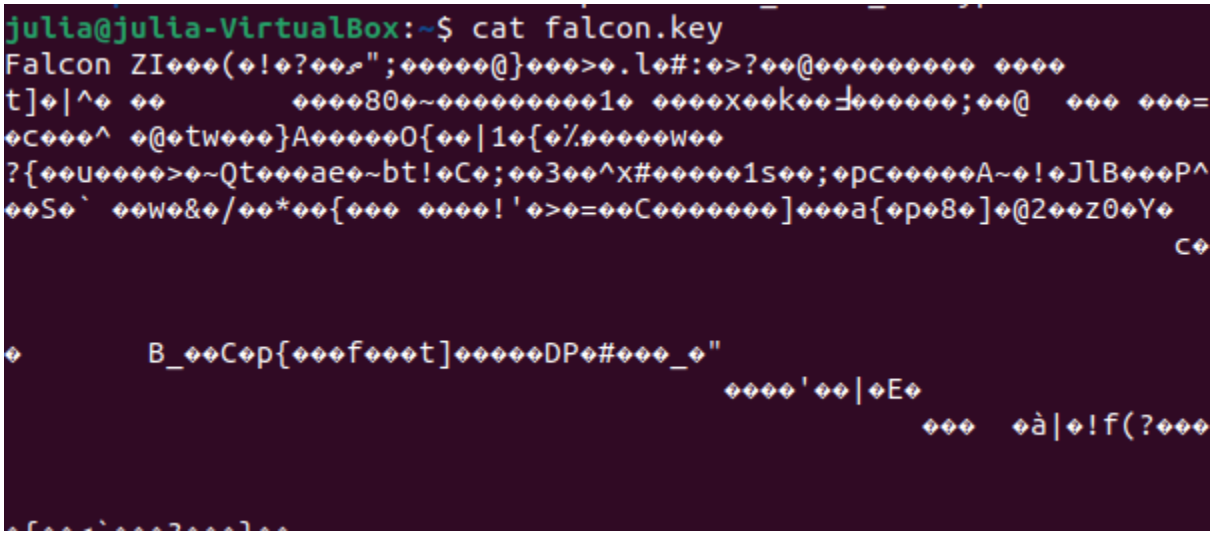
Obraz 3. Komunikat o poprawności wygenerowania kluczy

Film z generowania kluczy

https://drive.google.com/file/d/1H33TMOgNVZ6grCXeNL_X3XV4l4D8cpgz/view?resourcekey



Obraz 4 i 5: Wyjątki: generowanie kluczy

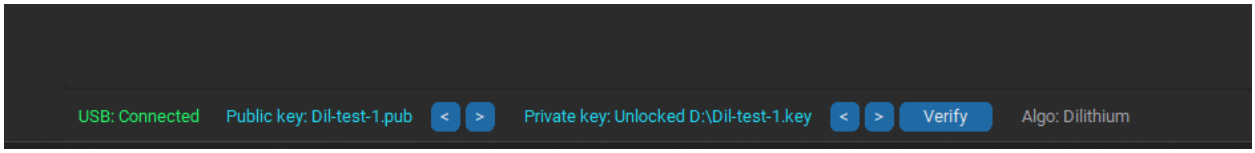


Obraz 6. Przykład klucza prywatnego Falcon

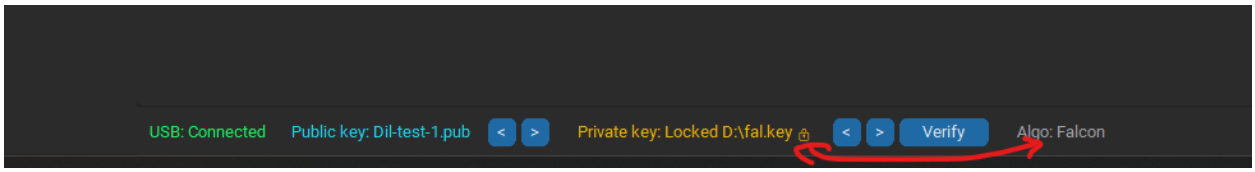
Wgranie kluczy z pamięci USB

Po podłączeniu przenośnej pamięci USB do systemu użytkownik ma możliwość użycia kluczy kryptograficznych zapisanych na tym nośniku. Aplikacja automatycznie wykrywa podłączone urządzenia USB i wyświetla dostępne klucze bez konieczności ręcznego wskazywania ich lokalizacji.

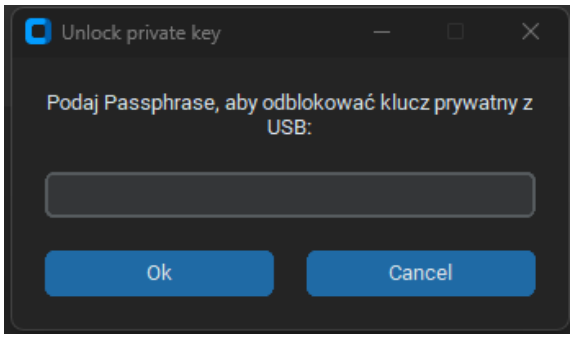
Wykrywanie realizowane jest przez mechanizm cyklicznego sprawdzania systemu co 3 sekundy, który analizuje podłączone nośniki wymienne oraz wyszukuje na nich pliki z kluczami o rozszerzeniach *.pub oraz *.key. Po odnalezieniu klucza aplikacja identyfikuje użyty algorytm i prezentuje go użytkownikowi, a w przypadku wykrycia wielu kluczy umożliwia wybór odpowiedniego pliku za pomocą przycisków nawigacyjnych. Cały proces wykrycia klucza na nośniku pokazany jest na obrazach 7-10



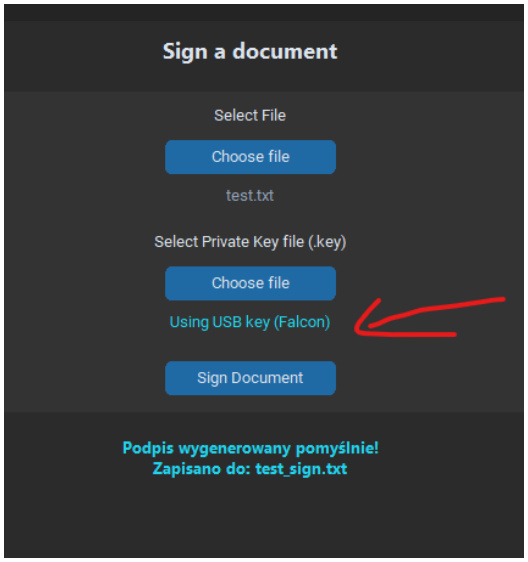
Obraz 7. Aplikacja wykryła klucz w podłączonej pamięci USB.



Obraz 8. Aplikacja wykryła kolejny klucz w pamięci USB.



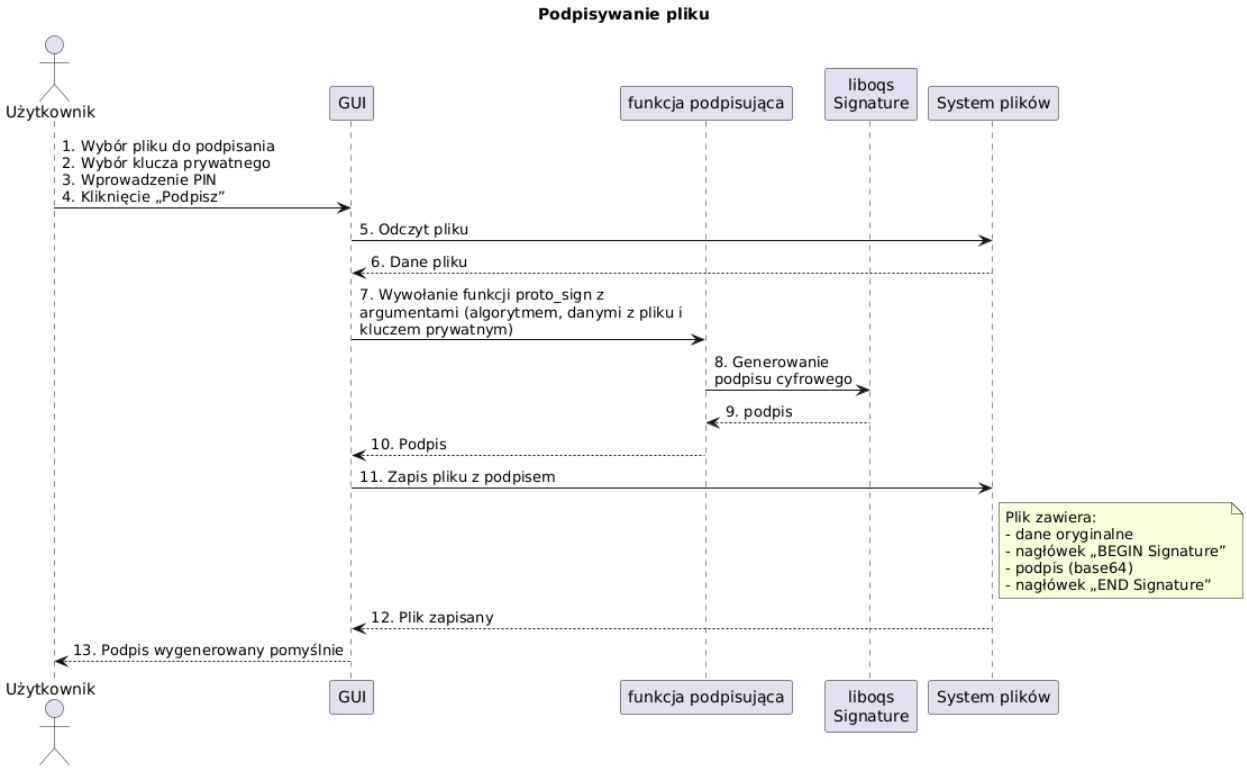
Obraz 9. System prosi o podanie hasła do klucza prywatnego (PIN).



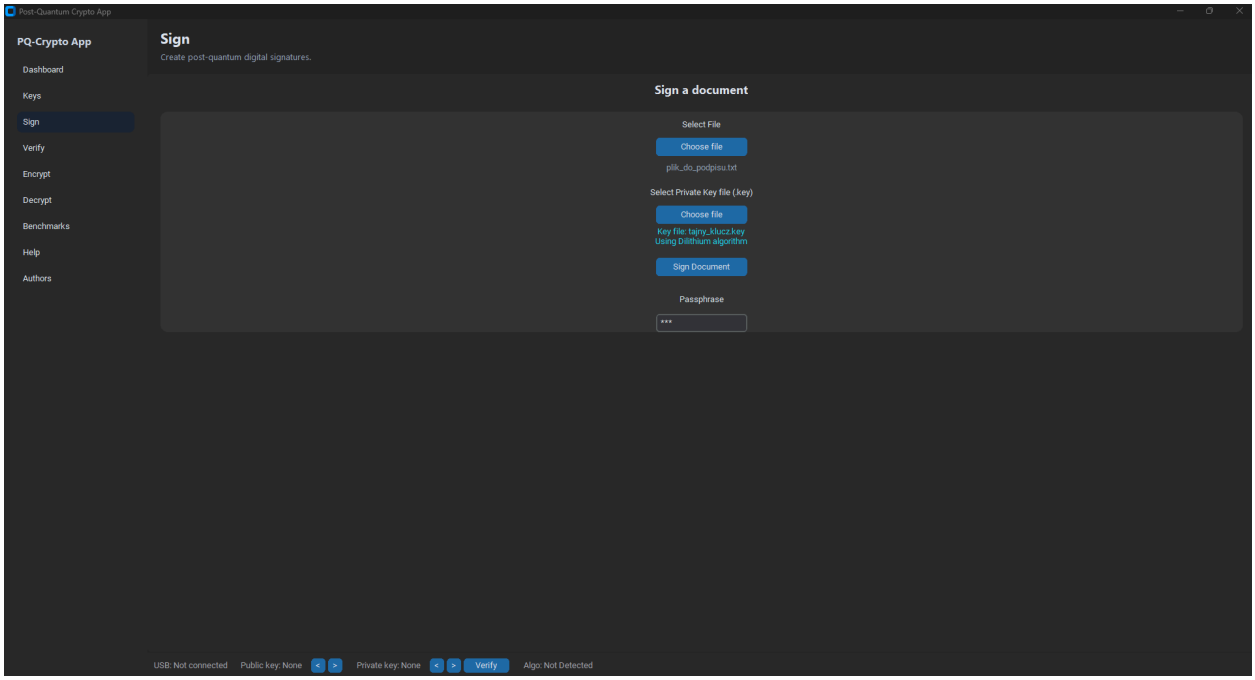
Obraz 10. Użycie klucza z pamięci USB do podpisania pliku.

Podpis i weryfikacja dokumentu

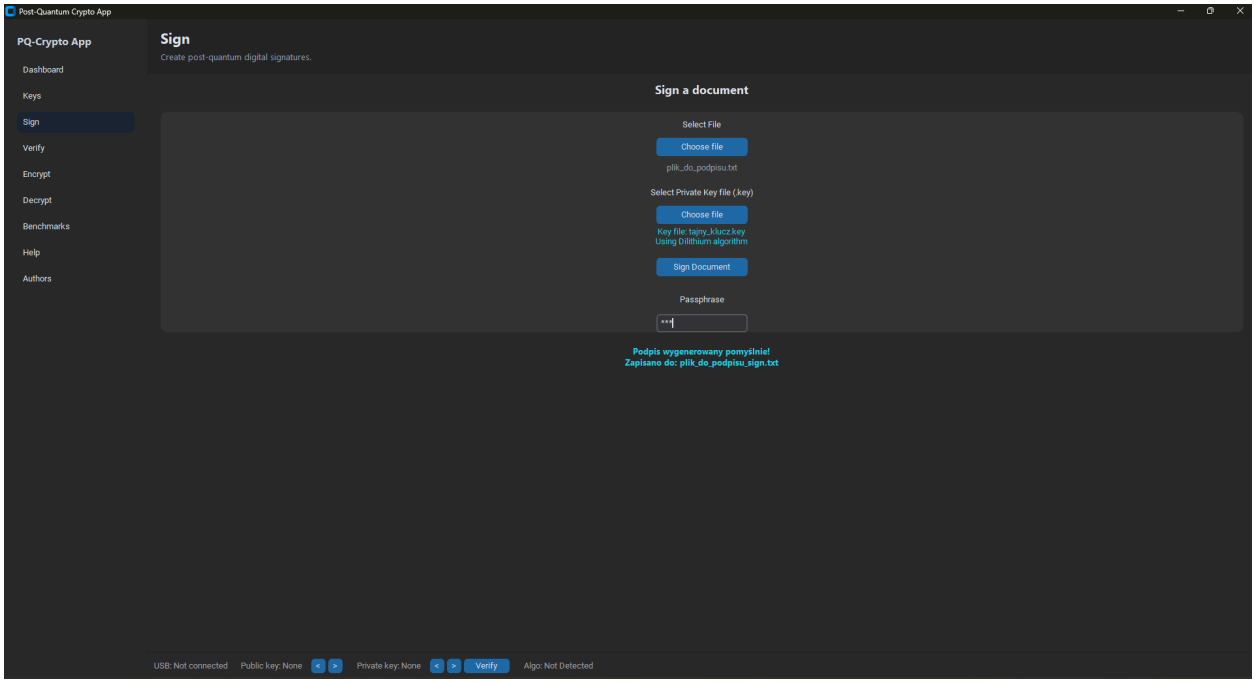
Diagram sekwencji – podpisywanie pliku



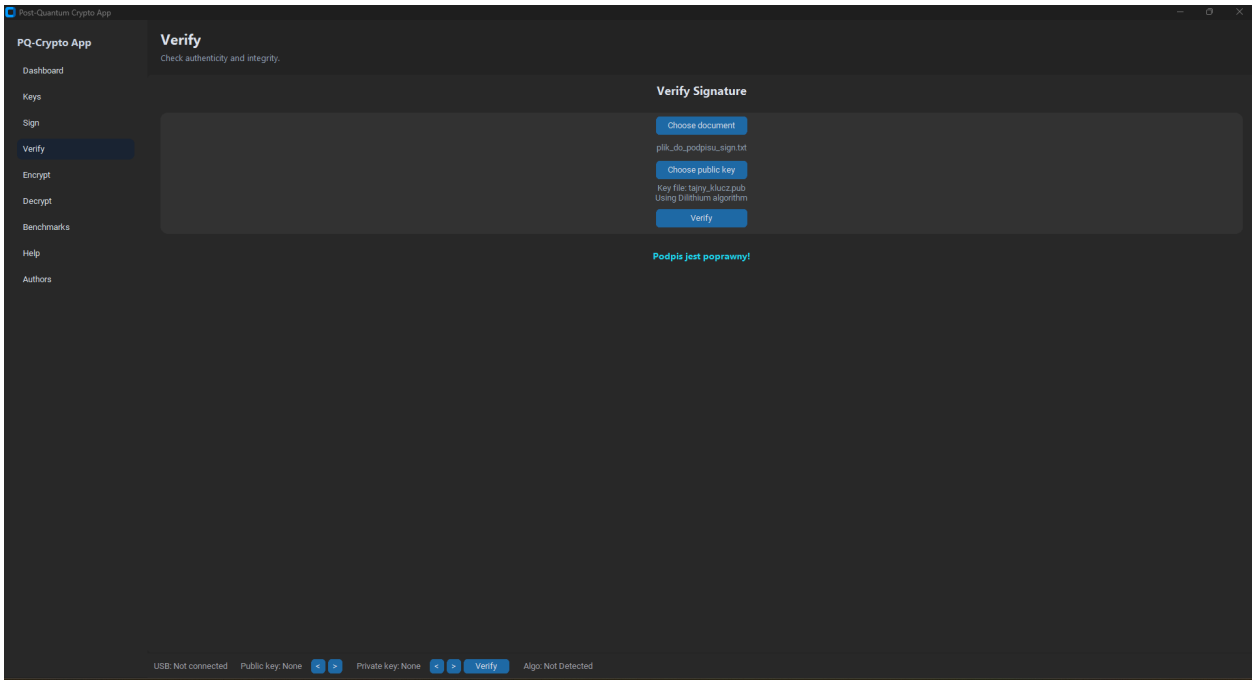
W celu podpisania dokumentu użytkownik wybiera plik, który chce podpisać, wskazuje klucz prywatny, wprowadza PIN, a następnie klika przycisk „Podpisz”. Wywoływana jest funkcja *proto_sign()*, po czym generowany podpis cyfrowy dołączany jest na końcu dokumentu wraz z nagłówkiem „==========Begin <nazwa algorytmu> Signature==========” oraz znacznikiem zakończenia podpisu „==========End <nazwa algorytmu> Signature==========”. Dzięki zastosowaniu nagłówka podczas weryfikacji podpisu wystarczy, aby użytkownik wskazał plik oraz odpowiadający mu klucz publiczny — nie ma potrzeby ręcznego podawania nazwy algorytmu. Podpisany dokument przedstawiono na obrazie 14. Proces podpisywania dokumentu od strony interfejsu graficznego zaprezentowano na obrazach 11 i 12, natomiast weryfikację podpisu pokazano na obrazie 13.



Obraz 11. Podpisywanie dokumentu



Obraz 12. Komunikat z poprawnym podpisaniem dokumentu.



Obraz 13. Weryfikacja podpisu.

Open
hello_world_sign.txt
~/
Save

```
Hello, World!


==========Begin Falcon Signature==========
OrivPKSHE0vDQJTu23IAgXofZZH4RjmR8KnfjUwu0ofcmPcInCK2tpfFmWEYg+uwRhblkbnQ6zqxWeNnmzKxM5TiTt8exI4Bi
Yyn6cQ1i8R8KpySwzt5UK2LMRb6ZSDS5KVtwiGXXptFfeiuLQI1r/
xuEHdIixueU8KE9t38ZH02dZcBwdnoSKnBS8yj55HJtL0EmbzPF0raLoS6XF9zWsd+I+WQSzJbKpRBomzSPiwb4Txq1KxRl07
5VnnXBm9Rjy3JKPZL2CWy0t48O1ND/7tU7wy+PYlAMQiW1ZjJLHCNp0Gtp5u4LNX5Q5vo1sVL8//
VfXRtAimZWLpbmVnlz696jclEyH3jOyDDpQoHXkVUsWS2Dwsi6WmNNXJ1FpG3lE6pN3ImnId+7m3/
f+zrIzLmxroE9Dsy5o1856/4szky+eDWHXKuwcdh42grGKVyVNoxRtJ+e9Qbs1WtsvU9wtDLT7Vm0MOce52BYTRHAVSes33GB
ME/Fn16p9ymR528qpDSzJh+LuNj46mYxlm/g3JdM713h6KI+eOKJIwx093mN66me5uDijIwKXwJ100kSX/
h+N4llHXbl7m7cqDVLO5Xl/
Kx63Bf1VNZpT+qNB3RhF9pUiqzNYhwp1GjHDmW9VtcnGTVLCkAWGd05JIe0DUwq26laVj8f17kQ01cpiG8phqbSdFnZRnWkk8
K/TCDsM6NG5N1wzhra2HT/
EBUSbQ1e99ZnA127ZRGJqUXj46NcEnvszeh+jL732mTl3ywCuf+k1PaT3KGbabKxIhjI3VLmukiOt3FvKqGoYv9V9EmBjBLkD
uycqs0zBSKZj28Zi4XCnp/
9Caxft5VqfE7duvF9etDX0l5z5shFCm7LZM7LeJCYXBOPEKm4nKmx+Znd5HjGZSfg2g+2PQ1TmemyHKO0nVfPvVMwa9ZGDlu6
Xk0xtUFNzJnIrcTXvz5M8vU5JKLLp2JXKKQpFTSo3KZOl/L1uweCLPThod/W7mdFQZZKZRLx/
JGwi4umhhApeT3MLrdriwyTWqkJsttOgzXNlb4k4GKKFHPjmioNY8RaNa02FsSkqoiaUN+U3rzWsq3Rke31hlPF1LYm32cVge
w39/
a9FJ53yZ76n4nIzBSLPb2vfpiIwxzJvuYJzKOcT+seHjufU9U1UBfnsm2ac1s8s9yuLazzDyvuznfapwyAMth91bbBBLxBUAz
61Cm/S8hCkIMS734X7frDsHyW/lkPYbWICL64lEpNQ48=
==========End Falcon Signature==========
```

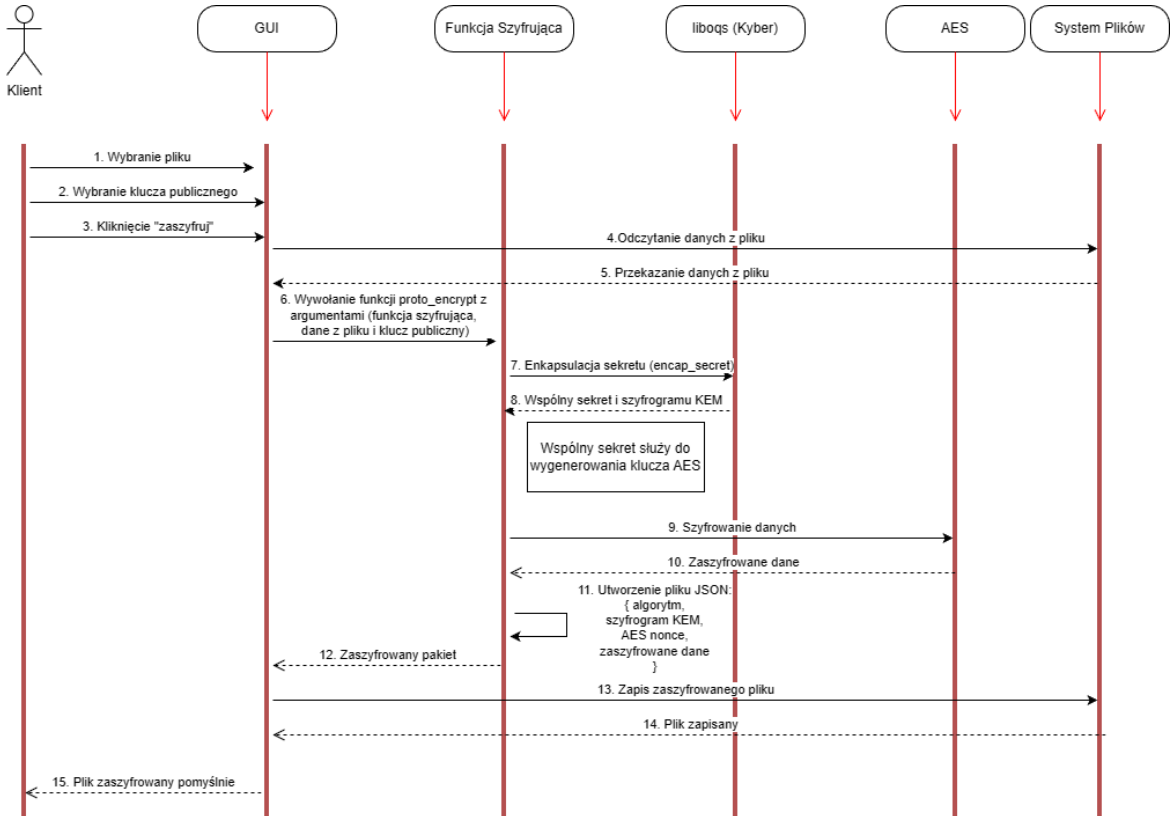
Obraz 13. Podpisany dokument za pomocą algorytmu Falcon.

Film z podpisu dokumentu

https://drive.google.com/file/d/1lh9L5X2Ml_JiH47w9mzf4B6AXiSPgW4k/view?resourcekey

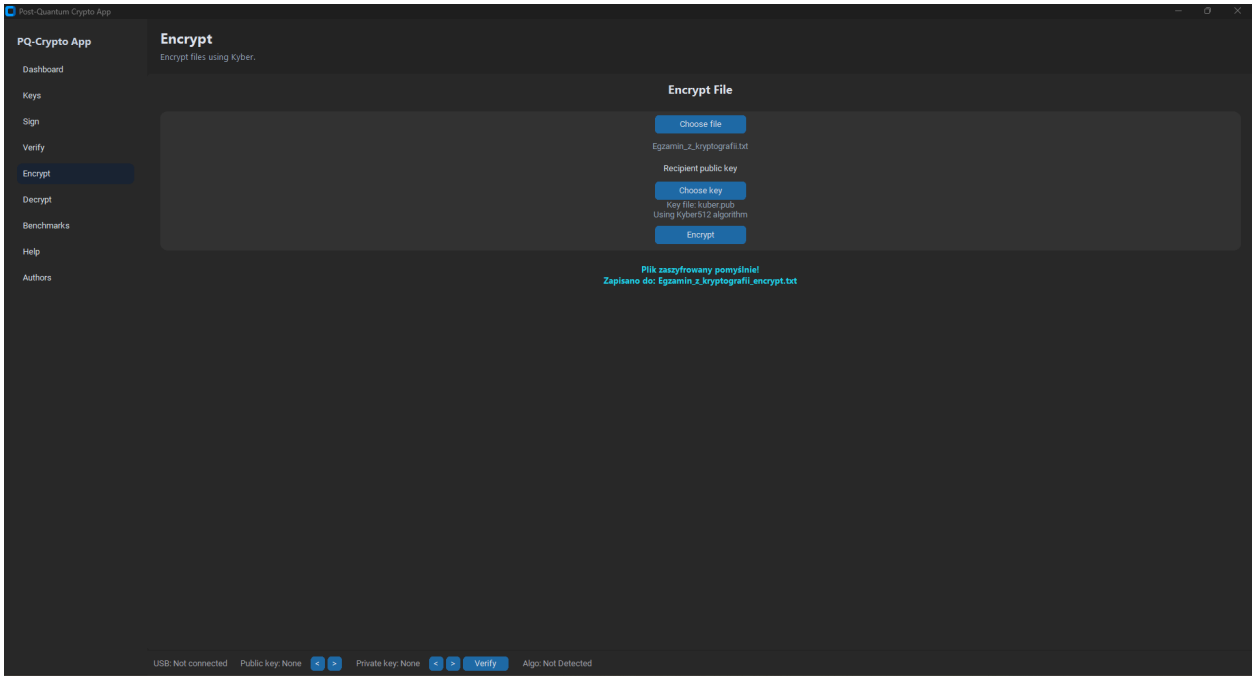
Szyfrowanie i odszyfrowanie

Diagram sekwencji – szyfrowanie (Kyber + AES-GCM)

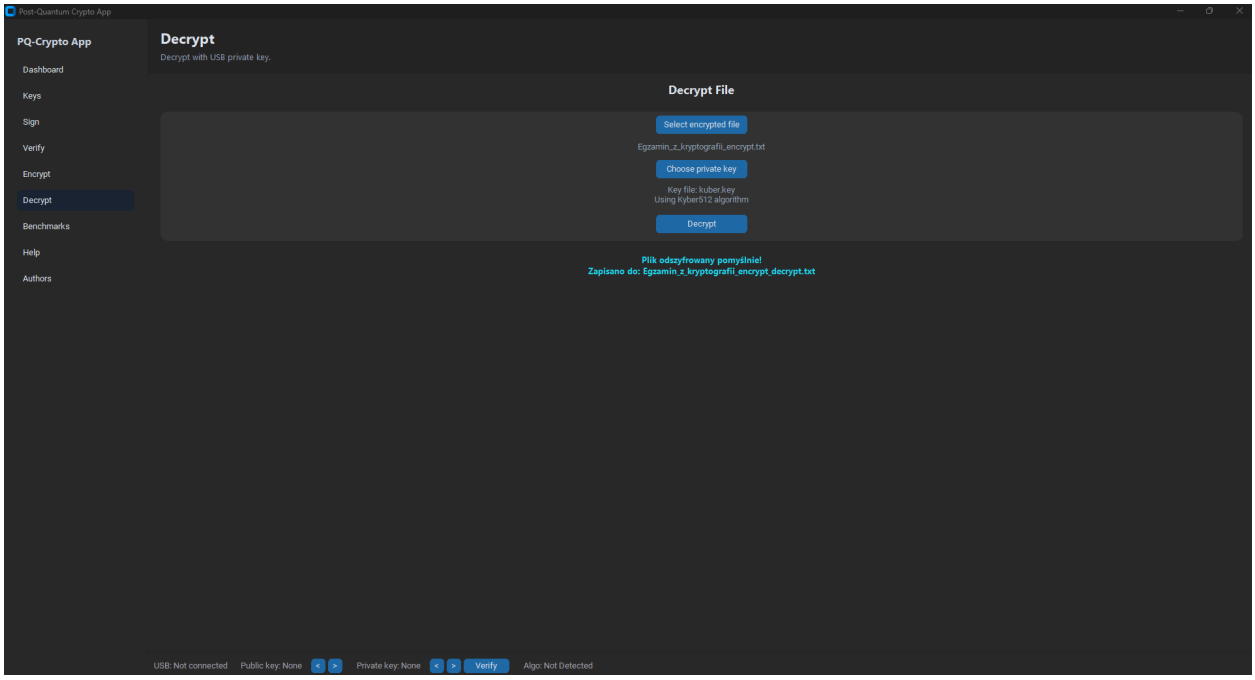


W celu zaszyfrowania pliku użytkownik wybiera plik do zabezpieczenia oraz klucz publiczny odbiorcy, a następnie klika przycisk „Encrypt”. Plik szyfrowany jest w sposób hybrydowy — algorytm postkwantowy Kyber wykorzystywany jest do bezpiecznego uzgodnienia wspólnego sekretu, natomiast właściwe dane pliku szyfrowane są symetrycznie przy użyciu algorytmu AES. Wynikiem operacji jest zaszyfrowany pakiet danych (json) zapisany w pliku, zawierający informacje o użytym algorytmie oraz wszystkie elementy niezbędne do późniejszego odszyfrowania. Proces szyfrowania pliku od strony interfejsu graficznego przedstawiono na obrazie 14.

W celu odszyfrowania pliku użytkownik wybiera wcześniej zaszyfrowany plik oraz odpowiadający mu klucz prywatny, który odblokowywany jest za pomocą PIN-u. Aplikacja weryfikuje zgodność algorytmu zapisanego w pliku z algorytmem klucza, a następnie przeprowadza proces deszyfrowania, ponownie wykorzystując algorytm Kyber oraz AES. Po poprawnym odszyfrowaniu użytkownik otrzymuje oryginalną treść pliku, którą może zapisać w wybranej lokalizacji. Proces deszyfrowania przedstawiono na obrazie 15.



Obraz 14. Szyfrowanie dokumentu



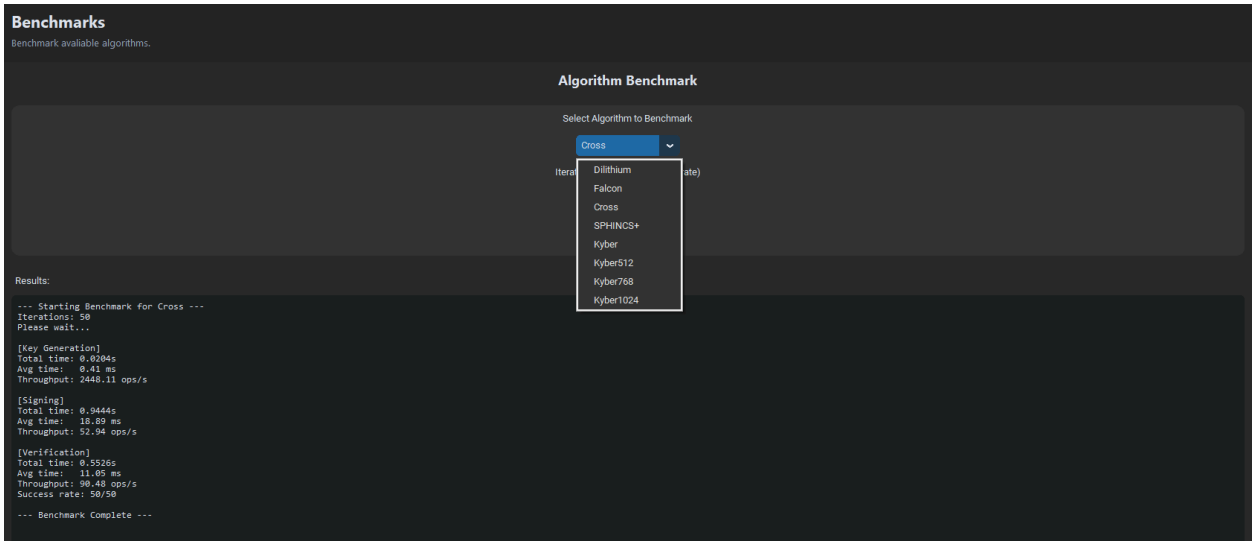
Obraz 15. Odszyfrowanie dokumentu.

Film z szyfrowania dokumentu

https://drive.google.com/file/d/17poelvJPU9q_fhadETrHixc7gCIF988I/view?resourcekey

Benchmark

Moduł benchmarku umożliwia porównanie wydajności wybranych algorytmów kryptograficznych obsługiwanych przez aplikację. Użytkownik wybiera algorytm oraz liczbę iteracji, po czym aplikacja automatycznie mierzy czas wykonania kluczowych operacji, takich jak generowanie kluczy, podpisywanie i weryfikacja lub szyfrowanie i deszyfrowanie danych. Wyniki prezentowane są w formie statystyk, które zawierają w sobie całkowity czas wykonania, średni czas pojedynczej operacji oraz przepustowość. Pozwala to w prosty sposób ocenić wydajność i porównać zachowanie algorytmów postkwantowych.



Obraz 16. Analiza wydajności algorytmu Cross za pomocą funkcji Benchmark.

Instalacja

Aby zainstalować nasz program, wymagane jest środowisko conda oraz python 3. Potrzebny jest również wrapper do biblioteki liboqs, czyli liboqs-python. Biblioteka ta nie jest jednak obecna w repozytorium pip ani conda, dlatego trzeba ją zainstalować i skompilować ręcznie. Można to zrobić na kilka różnych sposobów.

Sposób 1 - Instalacja poprzez liboqs-python

Mimo, że instrukcja mówi, że przed instalacją wrappera należy zainstalować najpierw bibliotekę liboqs. Natomiast nie jest wcale ona potrzebna do instalacji, ponieważ sam wrapper wykrywa czy liboqs jest zainstalowany w systemie. Jeżeli nie jest to sam go instaluje. Jest to bezpieczniejsza opcja, ponieważ instaluje ona minimalną wersję liboqs, potrzebną do pythonowego wrappera, a użytkownik nie musi wiedzieć jakie flagi przy kompilacji podać do kompilatora.

Sposób 2 - Ręczna instalacja liboqs

Przy kompilacji liboqs dostępnych jest wiele flag, które zmienią w pewnym stopniu działanie biblioteki, po jej skompilowaniu. Jeżeli więc potrzebujemy mieć kontrolę nad tymi flagami, to ten sposób jest wskazany ponad sposobem 1.

Domyślnie liboqs jest kompilowany jako biblioteka statyczna (liboqs.a), więc aby była ona biblioteką współdzieloną, należy użyć opcji ***-BUILD_SHARED_LIBS=ON***.

Niektóre kategorie kryptografii, nie są domyślnie włączone (LMS, XMSS). Są one jednak używane w naszym programie, dlatego aby były aktywne, należy podać flagi ***-DOQS_ENABLE_SIG_STFL_LMS=ON***, ***-DOQS_ENABLE_SIG_STFL_XMSS=ON***.

Generowanie kluczy dla powyższych kategorii, również nie jest aktywne po kompilacji, dlatego należy dodać flagę ***-OQS_HAZARDOUS_EXPERIMENTAL_ENABLE_SIG_STFL_KEY_SIG_GEN=ON***.

Istnieją również flagi, które pozwalają liboqs wykorzystywać większe zestawy instrukcji procesora takie jak AESNI, SSE, AVX512, SHA itd. Nie są one jednak wymagane, mogą one jedynie pozytywnie wpłynąć na wydajność kryptografii. Aby je aktywować należy skonsultować się z dokumentacją flag kompilacji. Przed dodaniem flag należy mieć na uwadze, że te zestawy instrukcji mogą nie być wspierane przez procesor, przez co biblioteka nie będzie działała.

Sposób 3 - Nieoficjalne repozytoria

Istnieją nieoficjalne repozytoria dla różnych systemów utrzymywanych przez użytkowników. Na systemie Arch Linux istnieje paczka liboqs. Instalacja z repozytorium skompiluje bibliotekę za nas. Ma ona ustawionych wiele flag, w tym te podane w sposobie 2.

Należy jednak zachować ostrożność, ponieważ te repozytoria są zarządzane przez użytkowników i nie są oficjalne, przez co instalowany przez nas liboqs może okazać się wirusem.

Instalacja liboqs-python

Po zainstalowaniu liboqs (lub nie, patrz sposób 1), należy zainstalować pythonowy wrapper do biblioteki. Aby to zrobić potrzebne jest utworzenie wirtualnego środowiska:

Linux i Mac

```
$ python3 -m venv venv
$ . venv/bin/activate
$ python3 -m ensurepip --upgrade
```

Windows

```
$ python3 -m venv venv
$ venv\Scripts\activate.bat
$ python3 -m ensurepip --upgrade
```

Następnie należy zainstalować wrapper:

```
$ git clone --depth=1 https://github.com/open-quantum-safe/liboqs-python
$ cd liboqs-python
$ pip install .
```

Na koniec można również uruchomić testy (opcjonalne):

```
$ python3 liboqs-python/examples/kem.py
$ python3 liboqs-python/examples/sig.py
$ python3 liboqs-python/examples/stfl_sig.py
$ python3 liboqs-python/examples/rand.py
$ nose2 --verbose
```

Używanie biblioteki

Wrapper nie jest zainstalowany w całym systemie, w kontekście użytkownika, ani w języku python. Żeby nasza aplikacja działała, należy dodać folder, w którym znajduje się wrapper do Ścieżki Pythona:

Linux i Mac

```
$ export PYTHONPATH=$PYTHONPATH:/path/to/liboqs-python
```

Windows

```
$ set PYTHONPATH=%PYTHONPATH%;C:\path\to\liboqs-python
```

Wybrane algorytmy

Kyber

Kyber jest w aplikacji jedynym algorytmem wykorzystanym do generowania kluczy kryptograficznych używanych do szyfrowania plików. Użytkownik ma do wyboru różne zestawy parametrów algorytmu Kyber: Kyber512, Kyber768, Kyber1024 odpowiadająca kolejnym poziomom bezpieczeństwa. Do właściwego szyfrowania używany jest symetryczny algorytm AES.

Podstawą algorytmu Kyber są modułowe kraty (Module-LWE/LWR) oraz problem Learning With Errors, który stanowi trudny problem obliczeniowy w przestrzeniach wielowymiarowych krat.

Algorytm jest zatwierdzony przez NIST i standaryzowany jako ML-KEM [Kyber].

Dilithium

Dilithium jest w aplikacji używany (tak jak i następne cztery algorytmy) do generowania kluczy i podpisów cyfrowych. Używamy jego wersji ML-DSA-44.

Podstawą algorytmu Dilithium są modułowe kraty (module lattices) oraz problemy kryptograficzne Module-LWE (Learning With Errors) oraz Module-SIS (Short Integer Solution). Konstrukcja opiera się na algebrze wielomianów nad pierścieniami ilorazowymi oraz na zastosowaniu Number Theoretic Transform (NTT) w celu przyspieszenia obliczeń. Bezpieczeństwo algorytmu wynika z trudności znalezienia krótkich wektorów w wysokowymiarowych kratach w obecności losowego szumu.

Algorytm Dilithium został zatwierdzony przez NIST i standaryzowany jako ML-DSA (FIPS 204). Jest on rekomendowany przez NIST jako główny post-kwantowy algorytm podpisu cyfrowego [Dilithium].

Falcon

W aplikacji używana jest wersja Falcon-1024. W porównaniu do Dilithium, Falcon generuje znacznie krótsze podpisy, kosztem bardziej złożonej i wrażliwej implementacji.

Podstawą matematyczną algorytmu Falcon są kraty typu NTRU, czyli kraty pierścieniowe oparte na arytmetyce wielomianów. Bezpieczeństwo algorytmu opiera się na problemie znajdowania krótkich wektorów w kratach NTRU, co jest problemem trudnym zarówno dla klasycznych, jak i

kwantowych komputerów. Falcon wykorzystuje również szybką transformatę Fouriera (FFT) do realizacji precyzyjnego próbkowania z rozkładu Gaussa w przestrzeni kratowej.

Algorytm Falcon został wybrany przez NIST jako alternatywny algorytm podpisu post-kwantowego i jest standaryzowany jako FN-DSA [Falcon].

Cross

W aplikacji używana jest wersja cross-rsdp-256-balanced. Nie jest on obecnie wykorzystywany jako standardowy mechanizm kryptograficzny, lecz służy do badań oraz testów przyszłych schematów podpisów post-kwantowych.

Algorytm Cross należy do rodziny schematów opartych na problemach algebraicznych i strukturach kratowych.

Algorytm nie został zatwierdzony przez NIST i nie posiada statusu standardu.

SPHINCS++

W aplikacji używana jest wersja SPHINCS+-SHAKE-128s-simple. Jest on algorytmem podpisu cyfrowego, który w przeciwieństwie do Dilithium i Falcon nie opiera się na kratach, lecz wyłącznie na bezpieczeństwie funkcji skrótu (hash functions) [Sphincs++].

Podstawą matematyczną algorytmu SPHINCS+ są drzewa Merkle'a, jednokrotne podpisy typu Winternitz (WOTS+) oraz konstrukcje haszujące odporne na kolizje i preimage attacks. Schemat jest stateless, co oznacza, że nie wymaga przechowywania stanu pomiędzy kolejnymi podpisami, co znacząco upraszcza jego użycie w praktyce.

Algorytm SPHINCS+ został zatwierdzony przez NIST i standaryzowany jako SLH-DSA (FIPS 205). W porównaniu z wcześniej omawianymi algorytmami, SPHINCS+ charakteryzuje się znacznie niższą wydajnością obliczeniową. Przeprowadzone testy wykazały, że wykonanie 50 iteracji zajmowało około 60 sekund, podczas gdy w przypadku pozostałych algorytmów czas ten wynosił około 1-2 sekund.