DOKUMENTACJA
Implementacja aplikacji do realizacji i weryfikacji podpisów cyfrowych

Wprowadzenie do cyberbezpieczeństwa – projekt

Mateusz Kowalczyk s188717, Szymon Lider s188850,
Mateusz Nowak s189420, Michał Tarnacki s188627

29 maja 2023

Spis treści

1 Wstęp

Dokumentacja dotyczy aplikacji do składania oraz weryfikacji podpisów cyfrowych. Przedstawimy w niej krótki opis czym są podpisy cyfrowe, jak działają, a później w jaki sposób zaimplementowaliśmy mechanizmy podpisów do aplikacji.

2 Wprowadzenie teoretyczne

2.1 Podpis cyfrowy

Podpis cyfrowy to kryptograficzny mechanizm zapewniający integralność, autentyczność i niezaprzeczalność dla elektronicznych dokumentów, wiadomości lub danych. Stanowi cyfrowy odpowiednik odręcznego podpisu, zapewniając niezmienność treści oraz potwierdzając tożsamość podpisującego.

Podpis cyfrowy w głównej mierze opiera się na kryptografii asymetrycznej. Proces ten wymaga wykorzystania pary kluczy: klucza prywatnego znanego tylko nadawcy oraz publicznego, który może być dowolnie rozpowszechniany. Klucz prywatny jest użyty do stworzenia podpisu, natomiast publiczny do weryfikacji autentyczności.

Utworzenie podpisu cyfrowego wygląda następująco:

1. Podpisujący za pomocą funkcji skrótu oblicza skrót (ang. *hash*) zawartości dokumentu.
2. Funkcja skrótu zwęża zawartość do ciągu o stałej długości.
3. Podpisujący szyfruje wartość funkcji skrótu przy użyciu klucza prywatnego, tworząc podpis cyfrowy.
4. Podpis następnie jest dołączany do dokumentu.

Cechy podpisu cyfrowego:

1. **Integralność:** podpis cyfrowy zapewnia, że treść podpisanego dokumentu nie zostanie zmieniona podczas przesyłu lub przechowania. Każda modyfikacja lub próba ingerencji w plik po podpisaniu unieważnia proces weryfikacji.
2. **Autentyczność:** podpis cyfrowy stwierdza autentyczność osoby podpisanej. Sprawdza, czy podpisany dokument pochodzi faktycznie od deklarowanego źródła bądź osoby. Jest to osiągnięte przy użyciu technik kryptograficznych, które łączą podpis z kluczem publicznym osoby podpisującej.
3. **Niezaprzeczalność:** podpis cyfrowy zapewnia niezaprzeczalność, w znaczeniu, że osoba podpisująca nie może zaprzeczyć swojego udziału w podpisywaniu dokumentu. Podpis cyfrowy jest mocnym dowodem na to, że podpisujący własnowolnie zatwierdził dokument lub plik, co uniemożliwia późniejsze wyrzeczenie się tego.

2.2 Algorytmy podpisów cyfrowych

- **RSA** - Algorytm Rivesta-Shamira-Adlemana. Jeden z pierwszych i obecnie najpopularniejszych asymetrycznych algorytmów kryptograficznych z kluczem publicznym
- **DSA** - Digital Signature Algorithm. Asymetryczny algorytm stworzony przez NIST w 1991 roku dla potrzeb DSS (Digital Sygnature Standard). Generowanie kluczy za pomocą tego algorytmu przebiega w następujący sposób:
  1. Generowana jest duża liczba pierwsza, zwykle oznaczana jako p, która służy jako parametr publiczny.
  2. Wybierana jest inna liczba pierwsza q taka, że q dzieli $p-1$. q jest mniejszą liczbą pierwszą i tworzy podgrupę w polu liczb pierwszych.
  3. Znajdowany jest element g w polu liczb pierwszych, który generuje cykliczną podgrupę rzędu q.
  4. Wybierany jest klucz prywatny x, który jest losową liczbą całkowitą z przedziału od 1 do $q-1$
  5. Obliczany jest odpowiadający mu klucz publiczny y, korzystając z równania
  $$y = g^x mod p$$
- **ECDSA** - Elliptic Curve Digital Signature Algorithm. Wariant DSA wykorzystujący ECC (Elliptic Curve Cryptography). Techniki te wykorzystują krzywe eliptyczne, które zapewniają bezpieczeństwo poprzez złożoność obliczeniową dyskretnych logarytmów na krzywych eliptycznych
- **EdDsa** - Edwards-curve Digital Signature Algorithm. Jest to schemat podpisu cyfrowego używającego wariant podpisu Schnorr bazującego na zakręconych krzywych Edwarda. Został zaprojektowany by był szybszy od poprzednich algorytmów, ale równie bezpieczny.

2.3 Funkcje skrótu

- **SHA-2** - Secure Hash Algorithm 2. Jest to zestaw kryptograficznych funkcji skrótu w skład którego wchodzi między innymi SHA-256. Realizowana w niektórych powszechnie używanych aplikacjach bezpieczeństwa i protokołach, w tym TLS i SSL, PGP, SSH, S/MIME, Bitcoin i IPsec
- **SHA-3** - Secure Hash Algorithm 3. Wyłoniona w 2012 roku w ramach konkursu ogłoszonego przez amerykański NIST. Charakteryzuje się wyższą wydajnością niż SHA-2. Algorytm SHA3-256 generuje 256-bitowy skrót. Ma architekturę „gąbki", bloki wejściowe są stopniowo „wchłaniane" w kolejnych etapach i mieszane z dużym rejestrem stanu. Blok wyjściowy jest

konstruowany w podobny sposób, przez „wyciskanie" kolejnych fragmentów danych wyjściowych z rejestru stanu, wielokrotnie go mieszając pomiędzy wyciskanymi blokami.

- **MD5** - Message-Digest algorithm 5. Algorytm będący popularną kryptograficzną funkcją skrótu, która z ciągu danych o dowolnej długości generuje 128-bitowy skrót. W 2004 roku znaleziono sposób na generowanie kolizji MD5, co obniża jego bezpieczeństwo do między innymi podpisywania plików

3 Aplikacja

3.1 Środowisko i język

Aplikację zaimplementowaliśmy w języku Python w wersji 3.11. Do jej tworzenia wykorzystaliśmy zintegrowane środowisko programistyczne PyCharm Community Edition 2022.1.1.

3.2 Biblioteki

Z powodu znaczącego skomplikowania algorytmów szyfrowania i deszyfrowania oraz wynikającego z niego niebezpieczeństwa popełnienia błędu przy ich własnoręcznej implementacji – w praktyce często należy stosować gotowe funkcje kryptograficzne udostępnione w bibliotekach. Z bogatej ich kolekcji dostępnej do wykorzystania w języku Python wybraliśmy następujące:

- PyCryptodome:
  - generowanie kluczy kryptografii asymetrycznej zgodnie z algorytmem DSA,
  - operacje zapisu i odczytu wspomnianych kluczy,
  - obliczanie wyniku funkcji skrótu SHA3-256,
  - tworzenie podpisu zgodnie z algorytmem DSA,
  - sprawdzanie podpisu zgodnie z algorytmem DSA;
- Cryptography:
  - szyfrowanie i odszyfrowywanie algorytmem AES.

Oprócz tego, do analizy i przetwarzania danych, skorzystano z:

- base64:
  - kodowanie i dekodowanie ciągów bajtów w sposób base64;
- xml:
  - tworzenie i przeglądanie dokumentów XML.

Ponadto do stworzenia interfejsu zostały użyte następujące pakiety

- CustomTkinter:
  - warstwa wizualna, biblioteka bazująca na Tkinter oferująca m.in obsługę motywów systemowych
- tkinterDnD:
  - interakcja z użytkownikiem poprzez przeciągnięcie pliku
- Pillow:
  - obsługa obrazów w aplikacji

3.3 Organizacja

Aplikacja jest podzielona na dwie części – właściwą i pomocniczą.

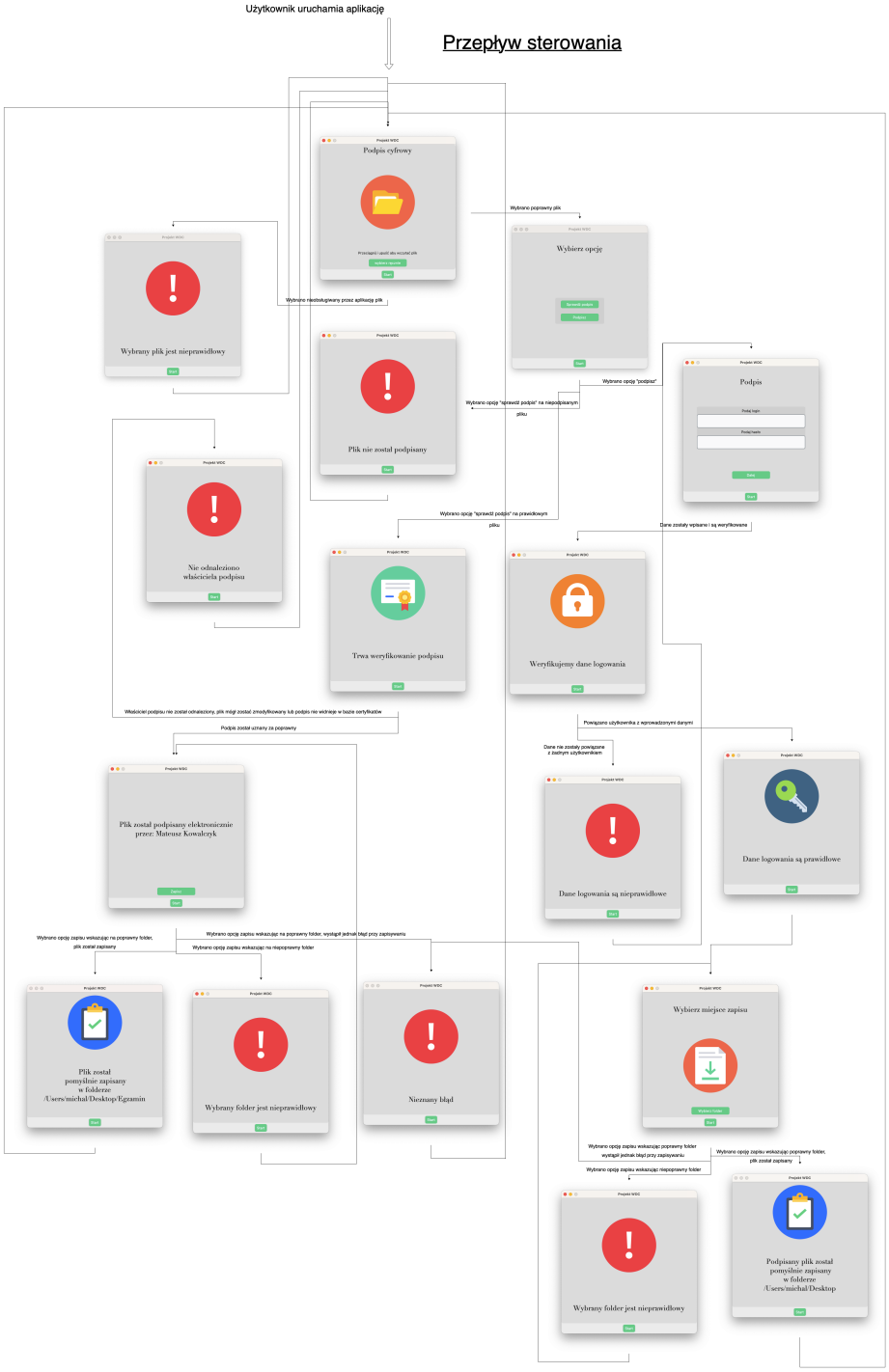
3.3.1 Część właściwa

Służy do składania i weryfikowania podpisów cyfrowych. Rozdzielona została na trzy pliki w sposób następujący:

- app_controller.py:
  - zawiera klasę AppController posiadającą kontrolery sterujące przepływem sterowania w aplikacji oraz gromadzącą dane wykorzystywane później do weryfikacji lub podpisywania plików
- views.py:
  - zawiera klasy rozszerzające Tk.Frame, a więc stanowiące jedynie formę prezentacji poszczególnych widoków między którymi przełącza się AppController
- controllers.py:
  - zawiera metody służące do podpisywania plików oraz weryfikacji ich podpisów

3.3.2 Przepływ sterowania

Wszystkie stany w jakich znajdować może się aplikacja zostały przedstawione na rysunku numer 1.



Rysunek 1: Przepływ sterowania.

3.3.3 Część pomocnicza

Służy jedynie do wygenerowania przykładowych kluczy i certyfikatów wykorzystywanych przez część właściwą. Znajduje się wewnątrz katalogu database_generating w pliku database_generating_scripts.py.

3.4 Interfejs użytkownika

3.4.1 Wstęp

Podczas pracy nad aplikacją staraliśmy się stworzyć interfejs przyjazny użytkownikowi, w którym intuicyjnie może się on poruszać. Wszelkie interakcje są na bieżąco weryfikowane, zatem ryzyko wystąpienia błędu zostało w miarę możliwości zminimalizowane. Sam interfejs zawiera szereg komunikatów przekazujących, jakie w danej chwili użytkownik może podjąć akcje.

3.4.2 Prawa autorskie

Wszystkie wykorzystywane grafiki pochodzą z serwisu Free Icon Shop, udostępnione zostały za darmo do użytku komercyjnego.

3.5 Przygotowanie kluczy i certyfikatów

Część pomocnicza aplikacji generuje przykładowe klucze i certyfikaty. Celem projektu nie jest jednak weryfikacja tożsamości osób, dla których się je wystawia. Działanie części pomocniczej polega jedynie na przygotowaniu danych, którymi część właściwa będzie operowała. Podczas korzystania z tej ostatniej należy założyć, iż certyfikaty zostały wystawione poprawnie po odpowiednim potwierdzeniu tożsamości osób ubiegających się o nie, a ich listę zapisano na lokalnym komputerze. Sposób generowania przykładowych kluczy i certyfikatów jest następujący:

1. Tworzony jest dokument XML do przechowywania certfikatów – metodami `xml.dom.minidom.Document`, `xml.dom.minidom.Document.createElement`, `xml.dom.minidom.Document.appendChild`.

2. Każdy dodawany użytkownik aplikacji posiada ustalone login, tożsamość (w postaci imienia i nazwiska) i hasło.

3. Dla każdego użytkownika generowany jest obiekt pary kluczy o długości klucza 2048 bitów – metodą `Cryptodome.PublicKey.DSA.generate`.

4. Do dokumentu XML dodawany jest certyfikat w prostym, autorskim formacie:

```
<Certificate>
   <Login>...</Login>
   <Identity>...</Identity>
   <PublicKey>...</PublicKey>
</Certificate>
```

gdzie znacznik `<Login>` zawiera login użytkownika, `<Identity>` – tożsamość użytkownika, natomiast `<PublicKey>` – klucz publiczny użytkownika w formacie PEM otrzymany z obiektu pary kluczy metodą `Cryptodome.PublicKey.DSA.DsaKey.exportKey`. Do uzupełniania dokumentu XML, oprócz funkcji wymienionych w punkcie 1., aplikacja wykorzystuje metodę `xml.com.minidom.Document.createTextNode`.

5. Klucz prywatny, który następnie zostanie zaszyfrowany, otrzymuje się w formacie PEM z obiektu pary kluczy metodą `Cryptodome.PublicKey.DSA.DsaKey.exportKey`. Klucz do jego zaszyfrowania to wynik funkcji skrótu SHA3-256 hasła użytkownika otrzymany metodą `Cryptodome.Hash.SHA3_256.new`. Do szyfrowania tworzy się obiekt szyfratora `cryptography.fernet.Fernet`, na którym wywołuje się metodę `cryptography.fernet.Fernet.encrypt`, szyfrującą klucz prywatny użytkownika algorytmem AES. Tak zabezpieczony klucz prywatny zostaje zapisany na dysku w pliku o nazwie w formacie *<login>_PrivateKey*

3.6 Składanie podpisów cyfrowych

Podpisywanie plików w omawianej aplikacji wykonywane jest w następujących krokach:

1. Użytkownik przesyła plik, który zamierza podpisać, oraz podaje swoje login i hasło.

2. Szukany jest plik z kluczem prywatnym użytkownika o podanym loginie. Podanie nieistniejącego loginu spowoduje nieodnalezienie pliku i przerwanie procesu w tym miejscu.

3. Klucz prywatny z odnalezionego pliku zostaje odszyfrowany algorytmem AES – przy użyciu metody `cryptography.fernet.Fernet.decrypt`. W tym celu wykorzystywany jest klucz stanowiący wynik funkcji skrótu (SHA3-256) hasła podanego przez użytkownika. Obliczany zostaje metodą `Cryptodome.Hash.SHA3_256.new`. Użyta funkcja odszyfrowująca automatycznie weryfikuje poprawność odszyfrowania danych i w przypadku odnalezienia błędu podnosi wyjątek `cryptography.fernet.InvalidToken`. Dzięki temu aplikacja potrafi stwierdzić, kiedy hasło wprowadzone przez użytkownika jest nieprawidłowe. Wówczas proces zostaje przerwany.

4. Odczytywany jest (w trybie binarnym) plik, który użytkownik wybrał do podpisu. Przy użyciu metody `Cryptodome.Hash.SHA3_256.new` obliczony zostaje wynik funkcji skrótu treści tego pliku.

5. Klucz prywatny użytkownika, dotychczas przechowywany w formie ciągu bajtów, zostaje wykorzystany do stworzenia obiektu klucza metodą `Cryptodome.PublicKey.DSA.import_key`.

6. Tworzony jest obiekt podpisu metodą `Cryptodome.Signature.DSS.new`. Funkcja ta jako argumenty otrzymuje obiekt klucza prywatnego używanego do podpisu oraz ciąg znaków wyznaczający tryb generowania sygnatury. Omawiana aplikacja wybiera tryb *FIPS 186-3*. Oznacza to, iż przy generowaniu podpisu wykorzystywane są liczby pseudolosowe, co poprawia jego bezpieczeństwo.

7. Skrót (ang. *hash*) treści podpisywanego pliku zostaje zaszyfrowany algorytmem DSA – metodą `Cryptodome.Signature.DSS.DssSigScheme.sign`. Tworzony jest w ten sposób ciąg bajtów stanowiący podpis cyfrowy.

8. Aplikacja generuje plik wyjściowy stanowiący dokument XMLw następującym formacie:

```
<?xml version="1.0" ?>
<SignedDocument>
   <FileContent>...</FileContent>
   <Signature>...</Signature>
   <SignerLogin>...</SignerLogin>
</SignedDocument>
```

   gdzie znacznik `<FileContent>` zawiera bajty stanowiące treść oryginalnego pliku zakodowane w base64, znacznik `<Signature>` – podpis cyfrowy zakodowany w base64, natomiast znacznik `<SignerLogin>` – login użytkownika, który podpisał plik.

9. Plik wyjściowy zostaje zapisany w lokalizacji podanej przez użytkownika.

3.7 Weryfikacja podpisów cyfrowych

Weryfikacja podpisu odbywa się następująco:

1. Użytkownik przesyła podpisany plik, który chce zweryfikować. Musi być to XML o odpowiedniej strukturze, inaczej nie zostanie poprawnie sparsowany.

2. Z pliku pobierana jest treść w postaci bajtowej, podpis oraz login podpisującego użytkownika.

3. Na podstawie loginu w bazie danych wyszukiwane są dane użytkownika oraz jego klucz publiczny. Jeżeli login nie istnieje w bazie, działanie programu kończy się w tym miejscu.

4. Treść pliku jest dekodowana z postaci base64, a następnie tworzony jest jej skrót (*hash*) za pomocą funkcji `Cryptodome.Hash.SHA3_256.new`. Otrzymany wcześniej klucz publiczny zostaje użyty do stworzenia obiektu klucza metodą `Cryptodome.PublicKey.DSA.import_key`.

5. Analogicznie jak przy podpisywaniu tworzony jest obiekt weryfikujący metodą `Cryptodome.Signature.DSS.new` w trybie *FIPS 186-3*. Tryb ten zapewnia niedeterministyczne generowanie liczb losowych.

6. Pobrany z pliku podpis zostaje zdekodowany z postaci base64.

7. Wywoływana jest metoda `Cryptodome.Signature.DSS.DssSigScheme.verify`, która jako argumenty przyjmuje skrót treści oraz zdekodowany podpis. Jeżeli podpis nie jest zgodny z treścią lub został podpisany kluczem należącym do innego użytkownika, zostaje zgłoszony wyjątek `ValueError`, którego wykrycie powoduje wyświetlenie stosownej informacji użytkownikowi. W przeciwnym wypadku program wyświetla dane użytkownika, który podpisał plik, i daje możliwość zapisania oryginalnej treści w wybranej lokacji.

3.8 Uruchomienie

W celu skorzystania z aplikacji należy:

1. uruchomić część pomocniczą – poprzez wywołanie skryptu znajdującego się w pliku database_generating/database_generating_scripts.py,

2. uruchomić część właściwą poprzez wywołanie skryptu znajdującego się w pliku main.py.

Wywołania skryptów można wykonać np. poprzez IDE, podając odpowiedni punkt wejścia do programu w konfiguracji uruchomienia.