Aplikacja do szyfrowania plików z użyciem kryptografii asymetrycznej

...

(temat 15)

Jan Dorniak 175959, Paweł Głomski 172026, Tomasz Rusinowicz 171872

Użyte narzędzia

Aplikacja została zaimplementowana w języku Python.

Do kryptograficznych zagadnień używaliśmy biblioteki PyCryptodome (fork biblioteki PyCrypto).

Do interfejsu graficznego użyliśmy biblioteki PyQt5.

Możliwości aplikacji (1) - generowanie kluczy RSA

Aplikacja umożliwia wygenerowanie pary kluczy: prywatnego i publicznego oraz zapisanie ich do pliku (z możliwością zabezpieczenia klucza prywatnego hasłem).

Możliwości aplikacji (2) - szyfrowanie plików

Aplikacja umożliwia szyfrowanie plików przy użyciu kryptografii asymetrycznej w połączeniu z kryptografią symetryczną. Aby zaszyfrować plik potrzebujemy klucza publicznego (wczytywanego z pliku) oraz pliku, który chcemy zaszyfrować. Wynikiem szyfrowania jest plik o rozszerzeniu “.jsonenc” (szczegóły implementacyjne na dalszych slajdach).

Możliwości aplikacji (3) - deszyfrowanie plików

Aplikacja umożliwia deszyfrowanie plików przy użyciu kryptografii asymetrycznej w połączeniu z kryptografią symetryczną. Aby odszyfrować plik potrzebujemy klucza prywatnego (wczytywanego z pliku), jego hasła (jeśli był podany przy tworzeniu) oraz zaszyfrowanego pliku w formacie “.jsonenc”, który chcemy odszyfrować. Wynikiem deszyfrowania (jeśli klucz prywatny jest prawidłowy) jest pierwotnie zaszyfrowany plik (szczegóły implementacyjne na dalszych slajdach).

Szczegóły implementacji (1) - Kryptografia symetryczna

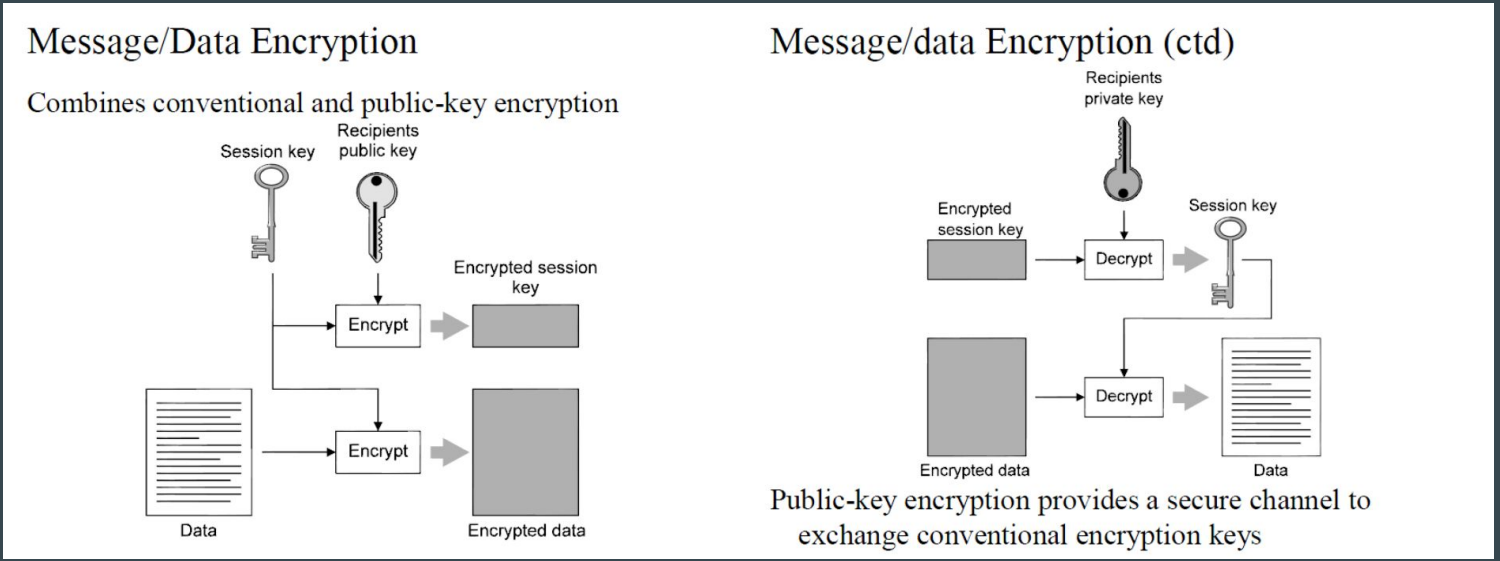
Szyfrowanie / deszyfrowanie plików odbywa się za pomocą symetrycznego szyfru blokowego AES.

Rozmiar klucza symetrycznego wynosi 256 bitów.

Do wyboru jest 8 różnych trybów szyfrowania:
CBC, CTR, CFB, OFB, CCM, EAX, GCM, OCB.

Szczegóły implementacji (2) - Kryptografia asymetryczna

Klucze asymetryczne (rozmiar 2048 bitów) wykorzystywane są do szyfrowania i deszyfrowania klucza symetrycznego, przy pomocy którego szyfrowane i deszyfrowane są pliki.



Szczegóły implementacji (3) - Wybór formatu

Początkowo planowaliśmy aby zaszyfrowane dane znajdowały się w formacie JSON razem z metadanymi.

Pojawiły się jednak problemy przy obsłudze dużych plików.

Problemy formatu JSON:

- ładowanie całego pliku do pamięci
- brak wsparcia dla przetrzymywania danych binarnych
  - wymaga kodowania szyfrogramu, co zwiększa czas pracy i rozmiar pliku
- brak wsparcia dla dopisywania danych (do istniejącego rekordu)

Szczegóły implementacji (4) - Stworzenie formatu

Aby rozwiązać ten problem stworzyliśmy własny format “jsonenc” (json + encrypted).

Jedynie sam nagłówek z metadanymi jest w formacie json.

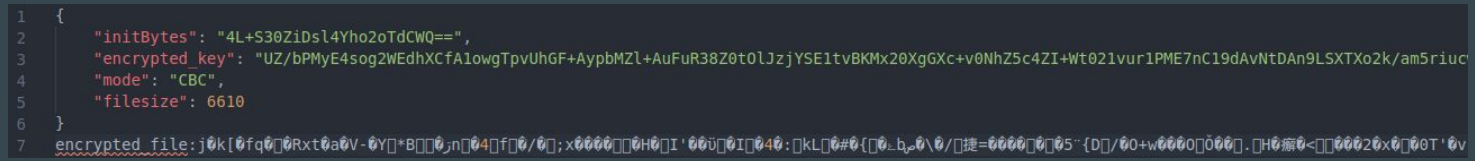
Zaszyfrowane dane pliku znajdują się zaraz za częścią nagłówkową.
Poprzedza je znacznik “encrypted_file:”, który pomaga ustalić początek szyfrogramu.

Szczegóły implementacji (5) - Format jsonenc

Nagłówek (JSON)	
Znacznik	**Opis**
initBytes	Bajty (nonce lub wektor inicjujący) używane do utworzenia obiektu szyfrującego
encrypted_key	Zaszyfrowany klucz symetryczny
mode	Tryb szyfrowania symetrycznego
filesize	Rozmiar zaszyfrowanych danych
Ciało	
Znacznik	**Opis**
encrypted_file	Plik zaszyfrowany symetrycznie

Szczegóły implementacji (6) - Plik jsonenc

Przykład pliku w formacie jsonenc:



Szczegóły implementacji (7) - Proces szyfrowania

Proces szyfrowania pliku:

1. Do pliku wyjściowego zapisywane są metadane w formacie JSON
2. Odczytywana jest część szyfrowanego pliku (do 64 kB)
3. Szyfrowanie części pliku
4. Dopisanie zaszyfrowanej części na koniec pliku wyjściowego
5. Jeśli to nie była ostatnia część pliku, idź do kroku 2

Analogicznie wygląda proces deszyfrowania.