OLSZTYŃSKA WYŻSZA SZKOŁA INFORMATYKI I ZARZĄDZANIA
im. Prof. Tadeusza Kotarbińskiego w Olsztynie
WYDZIAŁ INFORMATYKI I NAUK TECHNICZNYCH
Kierunek: Informatyka

Jakub Pastuszek

Bezpieczeństwo Systemów Komputerowych - Szyfrowanie plików wraz z przekazaniem klucza sesyjnego

Olsztyn 2013

Spis treści

Część I

Cel

Praca ma calu opracowanie aplikacji z graficznym interfejsem użytkownika umożliwiającej szyfrowanie i deszyfrowanie plików.

Aplikacja powinna wykorzystać wybrany algorytm szyfrowania pracujący w trybach CBC, ECB, OFB i CFB. W trybach OFB i CFB powinna umożliwić wybór długości podbloku. Dodatkowo powinna umożliwić wybór długości klucza.

Plik wyjściowy procesu szyfrowania powinien zawierać informacje umożliwiające dobór parametrów potrzebnych do jego rozszyfrowania.

Aplikacja powinna wykorzystać jednorazowy losowy klucz sesji do szyfrowania danych. Klucz ten powinien być zapisany w pliku wynikowym w postaci zaszyfrowanej przy pomocy podanego przez użytkownika hasła.

Do zaimplementowania aplikacji wykorzystałem język programowania Ruby[1] i jako wymagany algorytm szyfrowania wybrałem algorytm AES.

[1] http://www.ruby-lang.org

Część II

Struktura pliku wyjściowego procesu szyfrowania

1 Nagówek

Nagłówek jest zapisany w postaci tekstowej i jest oddzielony od szyfrogramu dwoma znakami nowej linii ($\backslash n \backslash n$). Zawiera no następujące informacje:

- nazwa algorytmu szyfrowania
- tryb pracy algorytmu szyfrowania
- długość klucza w bitach
- długość podbloku w bitach (jeśli różny od długości bloku algorytmu szyfrującego)
- wektor inicjujący zakodowany szesnastkowo (jeśli potrzebny dla danego trybu pracy)
- zaszyfrowany hasłem, tym samy algorytmem ale w trybie ECB oraz z wykorzystaniem paddingu klucz z sesji zakodowany szesnastkowo

Przykładowy nagłówek

```
cipher "BF"
mode "CFB"
key_size 256
sub_block_size 8
initialization_vector "3b5caafe7864a00b"
session_key "054b5d98c8a1d621a082cd45935d56a6ea913241d8d54fc741ea9c16fc627d3cf2f877ff306f9143"
```

Format nagłówka

Do zapisu i odczytu nagłówka wykorzystana jest biblioteka SDL[1].

[1] http://sdl4r.rubyforge.org

2 Szyfrogram

Szyfrogram jest zapisany binarnie bez żadnych dodatkowych modyfikacji zaraz po dwóch znakach nowej linii sygnalizujących koniec nagłówka.

Długość danych wejściowych

Przy wykorzystaniu trybu wymagającego dodania tak zwanego paddingu (CBC, ECB) stosowany jest on w formacie zdefiniowanym przez standard PKCS#5[1].

Aplikacja pracując w trybie nie wymagającym paddingu (CFB, OFB) zapisuje ostatni blok w odpowiednio skróconej formie.

[1] http://en.wikipedia.org/wiki/Padding_(cryptography)

Część III

Interfejs użytkownika

3
Wykorzystana technologia

Do budowy interfejsu użytkownika została zastosowana biblioteka Qt[1]. Jest to wieloplatformowa technologia umożliwiająca budowę przenośnych aplikacji z interfejsem graficznym. Dzięki zastosowaniu biblioteki qtbindings[2] możliwe jest skorzystanie z niej z poziomu języka programowania Ruby.

[1] http://qt-project.org
[2] https://github.com/ryanmelt/qtbindings

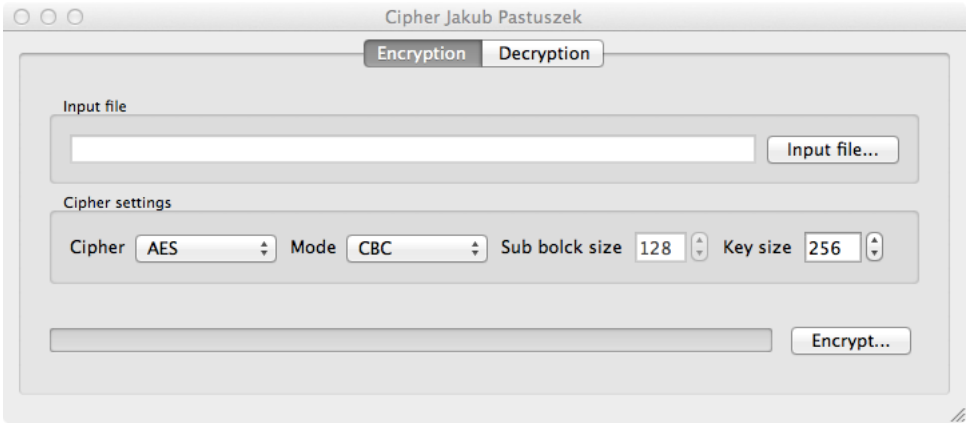
4
Funkcjonalność interfejsu

Szyfrowanie

Dzięki zastosowaniu biblioteki szyfrowania OpenSSL[1], która jest natywnie dostępna w Ruby, interfejs umożliwia wybór jednego z wielu dostępnych algorytmów szyfrowania.

Algorytmy te mogą pracować w różnych dostępnych trybach. Jednak by spełnić wymagania dla algorytmów umożliwiających tryb pracy ECB zaimplementowana została obsługa trybów OFB i CFB wspierająca wybór dowolnej długości podbloku. OpenSSL natywnie wspiera tryb OFB z podblokiem o długości 1 i 8 bitów dla wybranych algorytmów i będzie on wykorzystany zamiast dodatkowej implementacji gdy jest to możliwie.

Interfejs także umożliwia wybór długości klucza.



Rysunek 4.1: Główne okno aplikacji - szyfrowanie

Do wykonania operacji szyfrowania konieczny jest wybór pliku wejściowego.

Po naciśnięciu guzika *Encrypt...* interfejs zapyta o lokację i nazwę pliku wyjściowego oraz o hasło szyfrowania klucza sesji.

Okno wybory hasła szyfrowania oferuje opcję generowania losowego hasła. Dodatkowo jest możliwe odsłonięcia hasła.

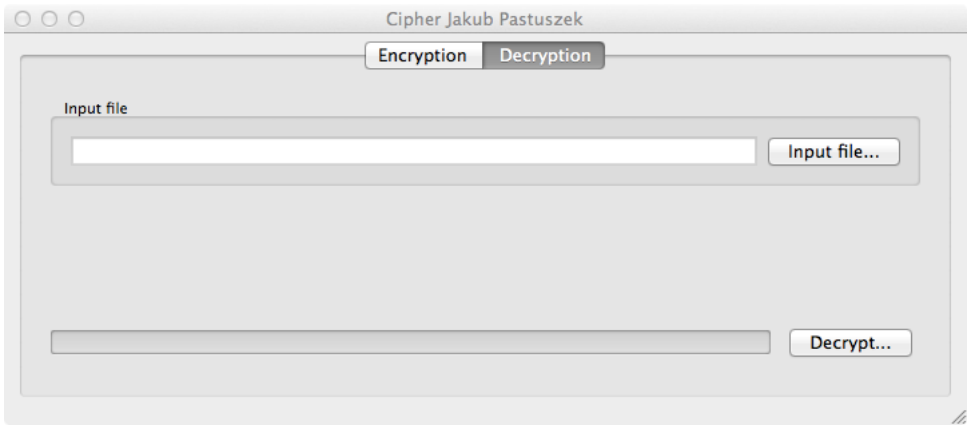
Po wyborze hasła rozpoczęty jest proces szyfrowania którego postęp jest przedstawiony za pomocą paska postępu.

[1] http://www.openssl.org

Rysunek 4.2: Okno wyboru hasła szyfrowania

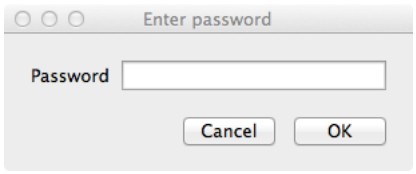
Deszyfrowanie

Okno deszyfrowania umożliwia wybór pliku zaszyfrowanego.



Rysunek 4.3: Główne okno aplikacji - deszyfrowanie

Po naciśnięciu guzika *Decrypt...* aplikacja zapyta o podanie hasła które będzie wykorzystane w procesie deszyfracji klucza sesyjnego.



Rysunek 4.4: Okno zapytania o hasło do deszyfracji

Po wprowadzeniu hasła aplikacja przystąpi do rozszyfrowywania danych. Postęp tej operacji będzie odzwierciedlony za pomocą paska postępu.

Część IV

Wyniki testów

5 Technologie wykorzystanie do testowania aplikacji

W procesie tworzenia tej aplikacji zostały zastosowane dwie technologie automatycznego testowania.

Klasy wchodzące w skład aplikacji są testowane za pomocą testów jednostkowych[1] wykorzystując bibliotekę RSpec2[2].

Podstawowa funkcjonalność aplikacji jest testowana w trybie linii komend przy zastosowaniu technologi Cucumber[3].

[1] http://en.wikipedia.org/wiki/Unit_testing
[2] http://rspec.info
[3] http://cukes.info

6
Wyniki testów

Wyniki testów jednostkowych

```
BlockCrypter::BlockSlicer
  should slice up input data into given size chunks

CipherSelector
  should provide list of available ciphers
  should raise error if selected cipher does not exist
  should provide ModeSelector for given cipher
  should provide flat list of all ciphers, modes and key lengths supported
  should privide block size
  ModeSelector
    should provide name of the cipher
    should provide list of available modes
    should raise error if selected mode does not exist
    should provide KeyLengthSelector for given mode
    should support selecting given mode
    should support selecting preffered mode
    #preferred_mode should select none mode when not available
    sub block selection
      should allow selecting sub block size from 8 to cipher block size in 8 bit increments
      should raise error if sub block size is not supported
      should raise error is sub block cannot be used with given mode
  KeyLengthSelector
    should provide name of the cipher
    should provide name of the mode
    should provide list of available key lengths
    should raise error if selected key lentght does not exist
    should provide CipherInfo for given key length
    should support selecting given key length
    should support selecting custom key length with supported cipher
    should preferr selecting predefined key length with supported cipher if requested matches
    should support selecting longest available key length
    #longest_key should select longest predefined key length when custom key length is supported
    #longest_key should select 256 key length for ciphers supporting only any key length
    #longest_key should select given key length for ciphers supporting only any key length
    should preferr selecting predefined mode for given sub block if available
    should use ECB preset for sub block preset is not available
```

```
    should allow using custom sub block processor over predefined preset when preffered

Encrypter
  should encrypt data stream with given cipher specs and key
  should provide initialization vector used
  should provide random initialization vector if not specified
  should provide nil initialization vector if not needed
  should work with custom key length
  should encrypt with padding
  should allow disabling of padding

Decrypter
  should decrypt data stream
  should work with custom key length
  should decrypt message without padding when padding is disabled
  custom sub block mode
    CFB
      encrypter
        should encrypt data stream with given cipher specs and key
      encrypter
        should decrypt data stream
    OFB
      encrypter
        should encrypt data stream with given cipher specs and key
      encrypter
        should decrypt data stream

Envelope::SDL
  should generate message
  should load streamed message

Filter
  should process input into output
  should allow passing multiple output values for one input
  should pass nil values
  may not produce value for input
  should output header before filtering
  should not output nil header value
  should output footer after filtering
  should not output nil footer value
  chaining
    should allow chaining of filters
    should call all header and footer handlers
  nesting
    should allow nesting many filters
    should call all header and footer handlers

IOEncrypter
  should encrypt input stream to output stream
  should use defferent session key for each run if not specified
  should use defferent initialization vector for each run if not specified

IODecrypter
  should decrypt input stream to output stream

SessionKey
  should be of given length in bits
  should be random
  can be encrypted
  can be encrypted (192)
```

```
  can be created from encrypted key

Finished in 0.37246 seconds
68 examples, 0 failures
```

Wyniki testów funkcjonalnych

```
Feature: Encrypting and decrypting streams with CLI application
  In ordre to be useful in shell cipher provides CLI application
  This application can take input stream and produce encrypted or decrypted stream

  Background:

  #Given cipher will print it's output
  #Given decipher will print it's output
  Scenario: Usage display
    Given cipher argument -h
    Given cipher is running
    When I wait for cipher termination
    Then cipher output should include 'Usage:'

  Scenario: Encrypt to stdout with AES256-CBC by default
    Given cipher is running
    When I write test.txt to cipher input
    Then I wait for cipher termination
    And cipher output should include following entries:
      | cipher "AES"          |
      | mode "CBC"            |
      | key_size 256          |
      | session_key           |
      | initialization_vector |

Feature: Encrypting and decrypting streams with different encryption parameters

  Background:
    Given cipher argument -p test
    Given decipher argument -p test
    Given content of test.txt file is used as cipher input

  @encryption @aes
  Scenario: Encryption with AES 128 ECB
    Given cipher argument -c AES -k 128 -m ECB
    When I run cipher with output sent through decipher
    Then decipher output should be the same as cipher input

  @encryption @aes
  Scenario: Encryption with AES 192 ECB
    Given cipher argument -c AES -k 192 -m ECB
    When I run cipher with output sent through decipher
    Then decipher output should be the same as cipher input

  @encryption @aes
  Scenario: Encryption with AES 256 ECB
    Given cipher argument -c AES -k 128 -m ECB
    When I run cipher with output sent through decipher
    Then decipher output should be the same as cipher input

  @encryption @aes
```

```
Scenario: Encryption with AES 128 CFB
  Given cipher argument -c AES -k 128 -m CFB
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes
Scenario: Encryption with AES 128 OFB
  Given cipher argument -c AES -k 128 -m OFB
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @native-sub-block @cfb
Scenario: Encryption with AES 128 CFB-1
  Given cipher argument -c AES -k 128 -m CFB-1
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @native-sub-block @cfb
Scenario: Encryption with AES 128 CFB-8
  Given cipher argument -c AES -k 128 -m CFB-8
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-to-native-sub-block @cfb
Scenario: Encryption with AES 128 CFB-8 (custom encryption, native decryption)
  Given cipher argument -c AES -k 128 -m CFB-8 -B
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-to-native-sub-block @cfb
Scenario: Encryption with AES 128 CFB (custom encryption, native decryption)
  Given cipher argument -c AES -k 128 -m CFB -B
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @native-to-custom-sub-block @cfb
Scenario: Encryption with AES 128 CFB-8 (native encryption, custom decryption)
  Given cipher argument -c AES -k 128 -m CFB-8
  Given decipher argument -B
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @native-to-custom-sub-block @cfb
Scenario: Encryption with AES 128 CFB (native encryption, custom decryption)
  Given cipher argument -c AES -k 128 -m CFB
  Given decipher argument -B
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-sub-block @cfb
Scenario: Encryption with AES 128 CFB-16
  Given cipher argument -c AES -k 128 -m CFB-16
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-sub-block @cfb
Scenario: Encryption with AES 128 CFB-64
  Given cipher argument -c AES -k 128 -m CFB-64
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input
```

```
@encryption @aes @custom-sub-block @cfb
Scenario: Encryption with AES 128 CFB-80
  Given cipher argument -c AES -k 128 -m CFB-80
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-to-native-sub-block @ofb
Scenario: Encryption with AES 128 OFB (custom encryption, native decryption)
  Given cipher argument -c AES -k 128 -m OFB -B
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-sub-block @ofb
Scenario: Encryption with AES 128 OFB-8
  Given cipher argument -c AES -k 128 -m OFB-8
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-sub-block @ofb
Scenario: Encryption with AES 128 OFB-16
  Given cipher argument -c AES -k 128 -m OFB-16
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @aes @custom-sub-block @ofb
Scenario: Encryption with AES 128 OFB-64
  Given cipher argument -c AES -k 128 -m OFB-64
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @bf
Scenario: Encryption with BF 128 ECB
  Given cipher argument -c BF -k 128 -m ECB
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @bf
Scenario: Encryption with BF 192 ECB
  Given cipher argument -c BF -k 192 -m ECB
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @bf
Scenario: Encryption with BF 192 CFB
  Given cipher argument -c BF -k 192 -m CFB
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @bf
Scenario: Encryption with BF 192 OFB
  Given cipher argument -c BF -k 192 -m OFB
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input

@encryption @bf
Scenario: Encryption with BF 192 CBC
  Given cipher argument -c BF -k 192 -m CBC
  When I run cipher with output sent through decipher
  Then decipher output should be the same as cipher input
```

```
25 scenarios (25 passed)
148 steps (148 passed)
1m40.415s
```