

Wojskowa Akademia Techniczna im. Jarosława Dąbrowskiego

Laboratorium z przedmiotu:
Wprowadzenie do Kryptologii

Sprawozdanie z ćwiczenia laboratoryjnego nr 2:
Funkcje skrótu

Prowadzący:
mgr inż. Marta Turowska

Wykonał: Radosław Relidzyński

Grupa: WCY20IY4S1

Data laboratoriów: 29.04.2021 r.

Spis treści

A.	Treść zadania	2
B.	Testowanie wektorów	2
C.	Generowanie skrótu dla pliku pdf	3
D.	Generowanie skrótu dla zmienionego pliku pdf	4
E.	Porównanie i wnioski	5

A. Treść zadania

1. Przetestować wektory pod kątem zwracania dobrego wyniku.
2. Wygenerować skrót dla pliku z prezentacją.
3. Zmienić 1 bajt w pliku.
4. Wygenerować skrót dla nowego pliku.
5. Wyciągnąć wnioski.

B. Testowanie wektorów

Korzystam ze zbioru wyników z repozytorium pod poniższym linkiem:

https://github.com/XKCP/XKCP/blob/master/tests/TestVectors/ShortMsgKAT_SHA3-224.txt

Do testowania będę używał polecenia:

```
printf "$message" | openssl dgst -sha3-224 -binary | xxd -p | tr a-f A-F
```

Lista kroków przy obliczaniu funkcji skrótu:

1. W miejsce *\$message* należy wpisać interesującą nas wiadomość.
2. *printf* konwertuje wiadomość na kod ASCII.
3. *openssl dgst -sha3-224 -binary* oblicza skrót przy użyciu algorytmu sha3-224 i zwraca go w zapisie binarnym.
4. *xxd -p* konwertuje do postaci hexadecymalnej.
5. *tr a-f A-F* zmienia litery na wielkie, żeby otrzymany wynik porównać z tym w pliku z repozytorium.

Testowane wartości (wraz z długością L):

1. Brak wiadomości (L = 0)
2. CC (L = 8)
3. 41FB (L = 16)
4. 1F877C (L = 24)
5. C1ECFDFC (L = 32)

Zwrócone wartości:

```
MINGW64:/c/Users/Radosław

Radosław@DESKTOP-TA2A43K MINGW64 ~
$ printf "" | openssl dgst -sha3-224 -binary | xxd -p | tr a-f A-F
6B4E03423667DBB73B6E15454F0EB1ABD4597F9A1B078E3F5B5A6BC7

Radosław@DESKTOP-TA2A43K MINGW64 ~
$ printf "\\xCC" | openssl dgst -sha3-224 -binary | xxd -p | tr a-f A-F
DF70ADC49B2E76EEE3A6931B93FA41841C3AF2CDF5B32A18B5478C39

Radosław@DESKTOP-TA2A43K MINGW64 ~
$ printf "\\x41\\xFB" | openssl dgst -sha3-224 -binary | xxd -p | tr a-f A-F|
BFF295861DAEDF33E70519B1E2BCB4C2E9FE3364D789BC3B17301C15

Radosław@DESKTOP-TA2A43K MINGW64 ~
$ printf "\\x1F\\x87\\x7C" | openssl dgst -sha3-224 -binary | xxd -p | tr a-f A-F
14889DF49C076A9AF2F4BCB16339BCC45A24EBF9CE4DCDCE7EC17217

Radosław@DESKTOP-TA2A43K MINGW64 ~
$ printf "\\xC1\\xEC\\xFD\\xFC" | openssl dgst -sha3-224 -binary | xxd -p | tr a-f A-F
A33C58DF8A8026F0F9591966BD6D00EED3B1E829580AB9BE268CAF39

Radosław@DESKTOP-TA2A43K MINGW64 ~
$ |
```

Sprawdzenie poprawności wyniku:

```
Python 3.10 (64-bit)
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # Dla ""
>>> "6B4E03423667DBB73B6E15454F0EB1ABD4597F9A1B078E3F5B5A6BC7" == "6B4E03423667DBB73B6E15454F0EB1ABD4597F9A1B078E3F5B5A6BC7"
True
>>> # Dla "CC"
>>> "DF70ADC49B2E76EEE3A6931B93FA41841C3AF2CDF5B32A18B5478C39" == "DF70ADC49B2E76EEE3A6931B93FA41841C3AF2CDF5B32A18B5478C39"
True
>>> # Dla "41FB"
>>> "BFF295861DAEDF33E70519B1E2BCB4C2E9FE3364D789BC3B17301C15" == "BFF295861DAEDF33E70519B1E2BCB4C2E9FE3364D789BC3B17301C15"
True
>>> # Dla "1F877C"
>>> "14889DF49C076A9AF2F4BCB16339BCC45A24EBF9CE4DCDCE7EC17217" == "14889DF49C076A9AF2F4BCB16339BCC45A24EBF9CE4DCDCE7EC17217"
True
>>> # Dla "C1ECFDFC"
>>> "A33C58DF8A8026F0F9591966BD6D00EED3B1E829580AB9BE268CAF39" == "A33C58DF8A8026F0F9591966BD6D00EED3B1E829580AB9BE268CAF39"
True
>>> _
```

Jak widać dla każdego wyniku zwracana wartość jest zgodna z oczekiwaną

C. Generowanie skrótu dla pliku pdf

Korzystam z okna openssl.exe

W ramach niego wywołam opcję:

sha3-224 \$file

gdzie w miejsce *\$file* wpiszę konkretny plik wraz ze ścieżką dostępu do niego

Zwrócona wartość:

```
D:\Program Files\Git\usr\bin\openssl.exe
OpenSSL> sha3-224 C:\Users\Radosław\Downloads\WKR2122L.pdf
SHA3-224(C:\Users\Radosław\Downloads\WKR2122L.pdf)= 01b2ac2433aff53afc462e95520e45a55d78dea89c44c5605f9d89ae
OpenSSL>
```

Przy pomocy zwykłej konsoli zamieniam litery na wielkie:

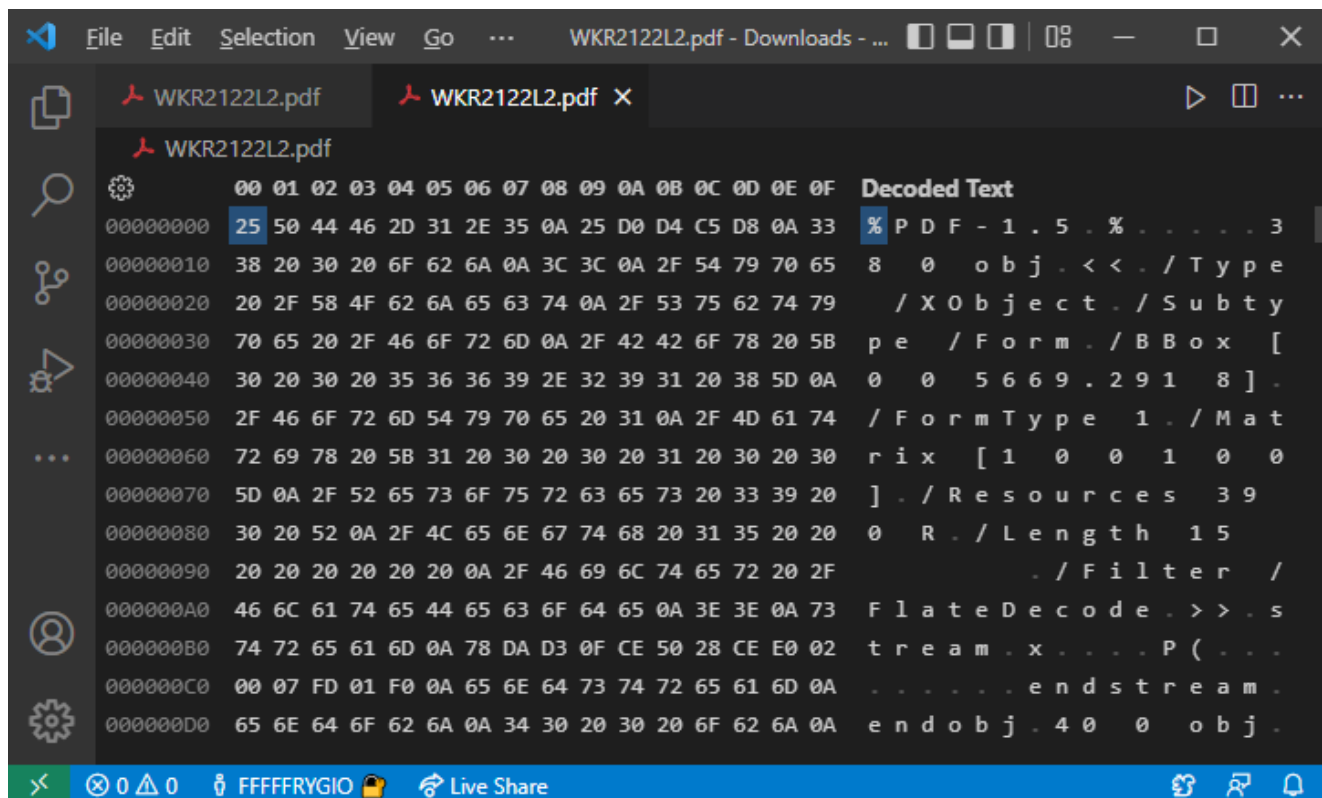
```
MINGW64:/c/Users/Radosław
Radosław@DESKTOP-TA2A43K MINGW64 ~
$ echo 01b2ac2433aff53afc462e95520e45a55d78dea89c44c5605f9d89ae | tr a-f A-F
01B2AC2433AFF53AFC462E95520E45A55D78DEA89C44C5605F9D89AE
Radosław@DESKTOP-TA2A43K MINGW64 ~
$
```

Otrzymany skrót:

01B2AC2433AFF53AFC462E95520E45A55D78DEA89C44C5605F9D89AE

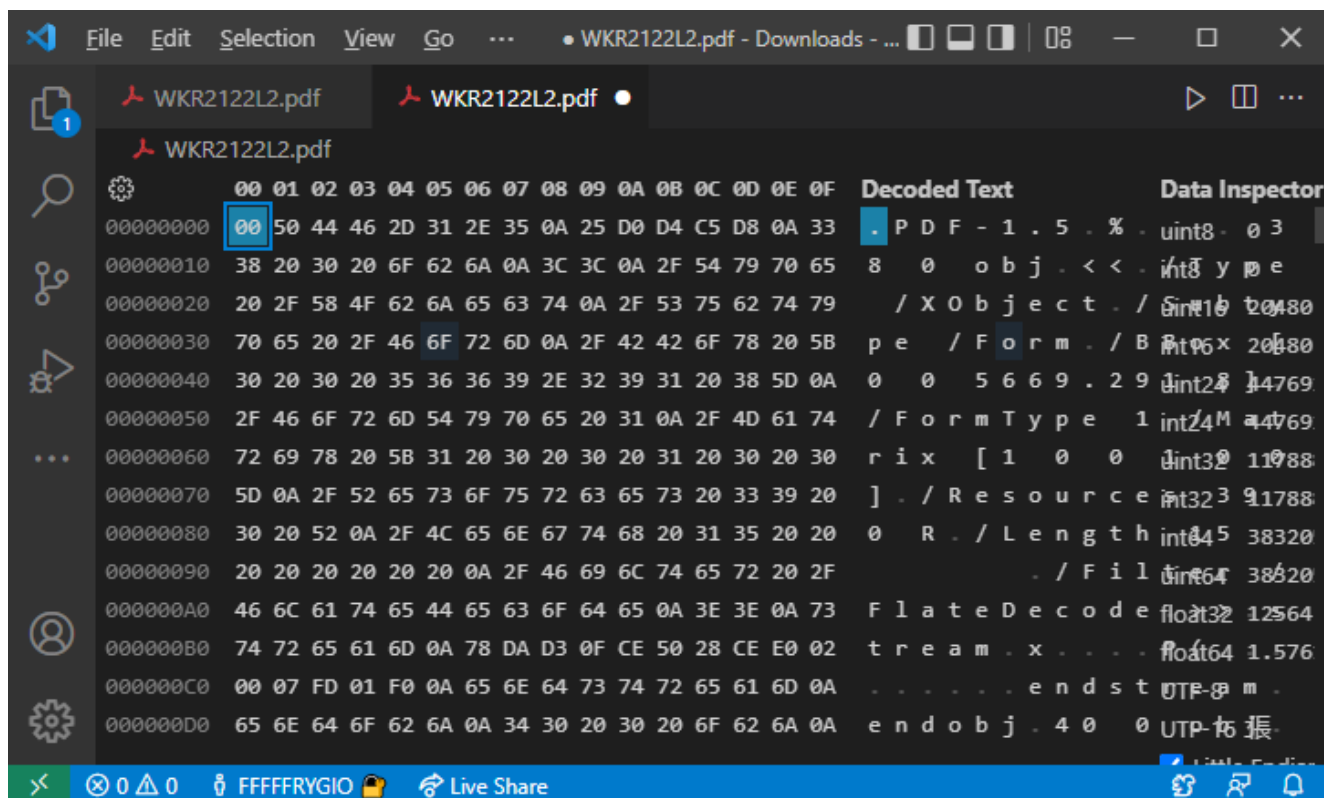
D. Generowanie skrótu dla zmienionego pliku pdf

Przy pomocy rozszerzenia „hex editor” otwieram skopiowany plik pdf i zmieniam w nim pierwszy bajt



The screenshot shows a hex editor window with the file 'WKR2122L2.pdf' open. The hex view displays the first 100 bytes of the file. The first byte, 0x25, is highlighted in blue. The decoded text view shows the corresponding ASCII characters: '%PDF-1.5'. The hex view is organized into columns of 16 bytes each, with offsets from 00000000 to 000000D0. The decoded text view shows the PDF header: '%PDF-1.5'. The status bar at the bottom shows '0 0 0', 'FFFFFRYGIO', and 'Live Share'.

Offset	Hex	Decoded Text
00000000	25 50 44 46 2D 31 2E 35 0A 25 D0 D4 C5 D8 0A 33	% P D F - 1 . 5 . % 3
00000010	38 20 30 20 6F 62 6A 0A 3C 3C 0A 2F 54 79 70 65	8 0 o b j . < < . / T y p e
00000020	20 2F 58 4F 62 6A 65 63 74 0A 2F 53 75 62 74 79	/ X O b j e c t . / S u b t y
00000030	70 65 20 2F 46 6F 72 6D 0A 2F 42 42 6F 78 20 5B	p e / F o r m . / B B o x [
00000040	30 20 30 20 35 36 36 39 2E 32 39 31 20 38 5D 0A	0 0 5 6 6 9 . 2 9 1 8] .
00000050	2F 46 6F 72 6D 54 79 70 65 20 31 0A 2F 4D 61 74	/ F o r m T y p e 1 . / M a t
00000060	72 69 78 20 5B 31 20 30 20 30 20 31 20 30 20 30	r i x [1 0 0 1 0 0
00000070	5D 0A 2F 52 65 73 6F 75 72 63 65 73 20 33 39 20] . / R e s o u r c e s 3 9
00000080	30 20 52 0A 2F 4C 65 6E 67 74 68 20 31 35 20 20	0 R . / L e n g t h 1 5
00000090	20 20 20 20 20 20 0A 2F 46 69 6C 74 65 72 20 2F	. / F i l t e r /
000000A0	46 6C 61 74 65 44 65 63 6F 64 65 0A 3E 3E 0A 73	F l a t e D e c o d e . > > . s
000000B0	74 72 65 61 6D 0A 78 DA D3 0F CE 50 28 CE E0 02	t r e a m . x P (. . .
000000C0	00 07 FD 01 F0 0A 65 6E 64 73 74 72 65 61 6D 0A e n d s t r e a m .
000000D0	65 6E 64 6F 62 6A 0A 34 30 20 30 20 6F 62 6A 0A	e n d o b j . 4 0 0 o b j .



Generuję skrót dla nowego pliku pdf:

```
D:\Program Files\Git\usr\bin\openssl.exe
OpenSSL> sha3-224 C:\Users\Radosław\Downloads\WKR2122L.pdf
SHA3-224(C:\Users\Radosław\Downloads\WKR2122L.pdf)= 01b2ac2433aff53afc462e95520e45a55d78dea89c44c5605f9d89ae
OpenSSL> sha3-224 C:\Users\Radosław\Downloads\WKR2122L2.pdf
SHA3-224(C:\Users\Radosław\Downloads\WKR2122L2.pdf)= dbc945e298dc7df33be810c34a45ede70870e7ac5103bdcabe681e10
OpenSSL>
```

Widać już, że znacząco różnią się od siebie

Przy pomocy zwykłej konsoli zamieniam litery na wielkie:

```
MINGW64:/c/Users/Radosław
Radosław@DESKTOP-TA2A43K MINGW64 ~
$ echo dbc945e298dc7df33be810c34a45ede70870e7ac5103bdcabe681e10 | tr a-f A-F
DBC945E298DC7DF33BE810C34A45EDE70870E7AC5103BDCABE681E10
Radosław@DESKTOP-TA2A43K MINGW64 ~
$
```

Otrzymany skrót:

DBC945E298DC7DF33BE810C34A45EDE70870E7AC5103BDCABE681E10

E. Porównanie i wnioski

Dla pewności zrobię porównanie dwóch skrótów i sprawdzę, czy są jakiegolwiek podobieństwa

```
Python 3.10 (64-bit)
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> s1 = "01B2AC2433AFF53AFC462E95520E45A55D78DEA89C44C5605F9D89AE"
>>> s2 = "DBC945E298DC7DF33BE810C34A45EDE70870E7AC5103BDCABE681E10"
>>> for i1, i2 in zip(s1, s2):
...     if i1 == i2:
...         print(i1, i2)
...
7 7
A A
>>> _
```

Tylko na 2 pozycjach są takie same znaki, co wskazuje na to, że są to 2 zupełnie różne od siebie ciągi znaków.

Jak więc widać, nawet 1 bajt różnicy powoduje, że openssl generuje w pełni różne wyniki.

Dobra funkcja haszująca posiada na tyle skomplikowany model kodowania informacji, żeby nie dało się w żaden sposób ocenić, jakie mechanizmy ma w sobie zastosowane.

Założmy scenariusz, że człowiek chcący odkryć model kodowania ma zbiór zwróconych wartości na podstawie zadanych tekstów. Na przykładzie różnic między skrótami pdf-ów można stwierdzić, że niezależnie od tego, czy teksty będą różniły się całym słowem, czy tylko literą, otrzymane wartości będą drastycznie inne i bez możliwości porównania ich. Uniemożliwia to znajdowania zależności do określania modelu kodowania.

Oczywiście nie zawsze model jest niewykrywalny. Źle opracowana funkcja haszująca może nie zapewniać bezpieczeństwa danych i może być możliwa do odkrycia, a czasami nawet możliwa do zastosowania wstecz (do dekodowania informacji).