

Implementacja szyfratorów strumieniowych oraz badanie ich właściwości

Paweł Koch 145330, 19.03.2022

1. Implementacja:

Do implementacji użyto języka Python w wersji 3.9.2 oraz 3.10 z bibliotekami: math, collections, time oraz random

BlumBlumShub:

Zaimplementowano podstawowy wariant algorytmu BlumBlumShub wykorzystując generowane automatycznie odpowiednio duże liczby Bluma

```
class BlumBlumShub(object):
    def __init__(self, seed=None):
        self.modulus = makeModulus()
        self.state = seed if seed is not None else randint(2, self.modulus - 1)
        self.state = self.state % self.modulus

    def seed(self, seed):
        self.state = seed

    def bitstream(self):
        while True:
            yield parity(self.state)
            self.state = pow(self.state, 2, self.modulus)

    def bits(self, n=20):
        outputBits = ''
        for bit in self.bitstream():
            outputBits += str(bit)
            if len(outputBits) == n:
                break

        return outputBits
```

Za pomocą powyższej implementacji wygenerowano 4 pliki z czego 3 zawierają po 20 000 bitów a jeden zawiera 1 000 000 wygenerowanych bitów.

Testowanie:

Testy zostały zaimplementowane zgodnie z podaną literaturą, tzn. standard FIPS 140-2, zostały one umieszczone w pliku `stest.py`, dla każdego wywołania informują one czy test został zakończony pomyślnie (PASSED) czy nie (FAILED) oraz sam wynik testu w postaci liczb np. wartości testu pokerowego.

Test pojedynczych bitów:

```
def bits_test(s: list):
    ones = s.count('1')
    if 9725 < ones < 10275:
        print(f"BITS TEST: PASSED, Value of test {ones}")
    else:
        print(f"BITS TEST: FAILED, Value of test {ones}")
```

Test serii:

```
def series_test(s: string):
    result_ones = []
    result_zeros = []
    for i in range(1, 26):
        result_ones.append(s.count(i * '1'))
        result_zeros.append(s.count(i * '0'))
    result_ones[6] = sum(result_ones[6:])
    result_ones = result_ones[:6]
    result_zeros[6] = sum(result_zeros[6:])
    result_zeros = result_zeros[:6]

    for i in range(6):
        result_ones[i] = result_ones[i] - sum(result_ones[i+1:])
        result_zeros[i] = result_zeros[i] - sum(result_zeros[i+1:])

    intervals = [(2315, 2685), (1114, 1386), (527, 723),
                 (240, 384), (103, 209), (103, 209)]
    result = 1
    for i in range(6):
        if not(intervals[i][0] < result_ones[i] < intervals[i][1]):
            result = 0
    if result == 1:
        print(
            f"SERIES TEST: PASSED, Value of test 1:{result_ones}, 0:{result_zeros}")
    else:
        print(
            f"SERIES TEST: PASSED, Value of test 1:{result_ones}, 0:{result_zeros}")
```

Test długiej serii:

```
def long_series_test(s: list):
    ones = s.count(26*'1')
    zeros = s.count(26*'0')

    if zeros == 0 and ones == 0:
        print(f"LONG SERIES TEST: PASSED, Value of test 1:{ones}, 0:{zeros}")
    else:
        print(f"LONG SERIES TEST: FAILED, Value of test 1:{ones}, 0:{zeros}")
```

Test pokerowy:

```
def poker_test(s: string):
    splitted = textwrap.wrap(s, 4)
    bits = {
        '0000': 0,
        '0001': 0,
        '0010': 0,
        '0011': 0,
        '0100': 0,
        '0101': 0,
        '0110': 0,
        '0111': 0,
        '1000': 0,
        '1001': 0,
        '1010': 0,
        '1011': 0,
        '1100': 0,
        '1101': 0,
        '1110': 0,
        '1111': 0,
    }
    for hex_number in splitted:
        bits[hex_number] += 1

    result = []
    for val in bits:
        x = bits[val]**2
        result.append(x)
    result = sum(result)
    result = 16/5000 * result - 5000
    if 2.16 < result < 46.17:
        print(f"POKER TEST: PASSED, Value of test {result}")
    else:
        print(f"POKER TEST: FAILED, Value of test {result}")
```

Dla wszystkich testowanych plików o długości 20 tysięcy bitów, generowanych z zaimplementowanego powyżej algorytmem BBS testy zostały ukończone pomyślnie co znaczy o prawidłowym działaniu algorytmu.

```

BITS TEST: PASSED, Value of test 9965
SERIES TEST: PASSED, Value of test 1:[4140, 679, 297, 156, 175, 158], 0:[4115, 786, 297, 225, 163, 146]
LONG SERIES TEST: PASSED, Value of test 1:0, 0:0
POKER TEST: PASSED, Value of test 15.94880000000012

```

```

BITS TEST: PASSED, Value of test 10023
SERIES TEST: PASSED, Value of test 1:[3956, 695, 242, 202, 168, 171], 0:[3981, 674, 257, 156, 163, 180]
LONG SERIES TEST: PASSED, Value of test 1:0, 0:0
POKER TEST: PASSED, Value of test 16.652799999999843

```

Algorytm został także poddany testom NIST i otrzymał następujący wynik:

RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES

C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	P-VALUE	PROPORTION	STATISTICAL TEST
0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	block-frequency	
0	0	0	0	0	0	0	1	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	1	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	1	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates
0	0	1	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	1	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	1	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	1	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	1	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates
1	0	0	0	0	0	0	0	0	0	-1.#IND00	0.0000 *	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	1	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	1	0	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	1	0	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates

[illegible]

[illegible]

0	0	0	1	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	1	0	0	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	1	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	1	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	1	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	1	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates	
0	0	0	0	1	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	1	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	1	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	1	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	1	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	1	0	0	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	1	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	1	0	0	0	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates	
0	0	0	0	0	0	1	0	0	0	0	-1.#IND00	1.0000	nonperiodic-templates
0	0	0	0	0	0	0	0	0	1	-1.#IND00	1.0000	nonperiodic-templates	
0	0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	overlapping-templates
0	1	0	0	0	0	0	0	0	0	0	-1.#IND00	1.0000	universal
0	0	0	0	0	0	0	0	1	0	-1.#IND00	1.0000	apen	
0	0	0	0	1	0	0	0	0	0	-1.#IND00	1.0000	serial	
1	0	0	0	0	0	0	0	0	0	-1.#IND00	1.0000	serial	
0	0	0	0	0	0	0	1	0	-1.#IND00	1.0000	linear-complexity		

The minimum pass rate for each statistical test with the exception of the random excursion (variant) test is approximately = 0.691504 for a sample size = 1 binary sequences.

For further guidelines construct a probability table using the MAPLE program provided in the addendum section of the documentation.

Co znowu jest odpowiednim wynikiem świadczącym o prawidłowości algorytmu

Szyfrator strumieniowy:

Zaimplementowany wyżej generator ciągu pseudolosowego użyto do implementacji szyfratora strumieniowego:

```
def encrypt(bits, key):  
    xored = [xor(a, b) for (a, b) in zip(bits, key)]  
    return xored  
  
def decrypt(bits, key):  
    xored = [xor(a, b) for (a, b) in zip(bits, key)]  
    return xored
```

Następnie zaszyfrowano wygenerowany plik o długości ~2500 bajtów czyli około 20 000 bitów zawierający Lorem ipsum (bardzo długie) przeprowadzając jednocześnie testy FIPS na zaszyfrowanej wiadomości. Otrzymano następujący wynik:

```
BITS TEST: FAILED, Value of test 10683  
SERIES TEST: PASSED, Value of test 1:[3848, 549, 51, -56, 61, 370], 0:[5241, 1856, 926, 92, 0, 0]  
LONG SERIES TEST: PASSED, Value of test 1:0, 0:0  
POKER TEST: FAILED, Value of test 5646.3936000000001  
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus sed suscipit lectus, et suscipit diam. Sed ligula purus, auctor in volutpat sed, sollicitudin et dolor. Vivamus quis maximus nunc, eget condimentum diam. Integer egestas, mi posuere euismod eleifend, diam eros volutpat arcu, ut fringilla leo metus a nisl. Sed quis dignissim ligula. Etiam aliquet elementum ligula eu bibendum. Sed vel erat sagittis, tincidunt lacus ut, gravida elit.  
  
Curabitur porta congue tellus. Proin vitae dui ut nibh pharetra iaculis et quis nibh. Nunc tincidunt ipsum a lacus euismod maximus. Suspendisse eu massa justo. Praesent aliquam neque luctus urna blandit, ut laoreet turpis elementum. Donec porta est a dignissim rhoncus. Vivamus finibus eu turpis ac malesuada. Phasellus hendrerit lacus ante, non mollis tortor tristique in.  
  
Donec sem odio, facilisis a nulla vel, malesuada tristique nisl. Sed sit amet felis vitae velit ornare tincidunt non nec lorem. Aenean nec odio justo. Vestibulum consectetur commodo orci, eget pulvinar odio placerat ut. In convallis accumsan eros, quis scelerisque metus sagittis sed. Suspendisse blandit vestibulum risus, non eleifend neque mattis quis. Nunc eu lectus arcu.  
  
Mauris leo nisl, dapibus vitae orci quis, cursus vulputate erat. Nam vulputate nibh sed ante congue, ac egestas ipsum varius. Aliquam erat volutpat. Nulla congue laoreet nisi, vestibulum varius ipsum interdum et. Ut eget tortor sit amet massa dignissim ultricies at sed sem. Nulla facilisi. Praesent varius tellus eu magna lobortis dictum. Nulla sit amet est mauris. Interdum et malesuada fames ac ante ipsum primis in faucibus. Pellentesque efficitur pretium ex non cursus. Mauris luctus, nibh a faucibus tristique, ante nulla malesuada ex, eu ornare metus arcu sit amet quam. Ut sit amet felis vel nulla feugiat aliquam. Phasellus ac diam velit. Nam laoreet congue augue, lobortis efficitur sapien mollis vel.  
  
Aliquam erat volutpat. Vestibulum iaculis rhoncus convallis. Proin porttitor magna neque, at hendrerit dui tempus non. Pellentesque sagittis, nunc vel congue malesuada, sapien dolor lacinia metus, id ullamcorper ante massa sed ligula. Vivamus dui mi, pretium nec velit non, fermentum interdum orci. Donec dictum, sem in maximus dignissim, erat ante eleifend est, sit amet vehicula lorem arcu et sapien. Morbi dictum neque vitae blandit tincidunt. Morbi sodales orci eget lacus tincidunt, varius sollicitudin nisl auctor. Vestibulum vestibulum enim in orci eleifend, at dictum ligula po
```

Testy zakończone porażką oznaczają, że zaszyfrowana wiadomość nie jest ciągiem pseudolosowym.

Wnioski:

Szyfratory strumieniowe nie należą do najbezpieczniejszych rozwiązań mimo pozorów. Są one narażone na wiele rodzajów ataków przez wykonywanie prostych operacji, dodatkowo posiadając klucz jesteśmy odszyfrować każdą wiadomość zaszyfrowaną danym kluczem co jest oczywiście słabością wymagająca dodatkowo bezpiecznego przechowywania klucza szyfrującego.