Лабораторная работа 3

Поточные криптосистемы

Вариант 18

Раконяц Даниела

7 группа

2023

Для выработки ключей криптосистем и гаммы нужно использовать криптографические генераторы псевдослучайных чисел. Необходимость отказа от обычных программных генераторов (например, от линейного конгруэнтного метода) обусловлена двумя причинами: незащищенностью реализации и возможностью предугадать значение. Выделяют класс криптографических генераторов, базовым элементом которых является регистр сдвига с линейной обратной связью (РСЛОС). При этом, как правило, требуется, чтобы РСЛОС генерировал линейную рекуррентную последовательность (ЛРП) с максимально возможным периодом (относительно *длины* *РСЛОС* (количества ячеек памяти у РСЛОС)), высокой линейной сложностью и хорошими статистическими свойствами. Так как ЛРП не обладает нелинейными свойствами, то возможно восстановление РСЛОС по небольшому отрезку ЛРП этого РСЛОС. Поэтому, для построения криптографических генераторов использует сочетание РСЛОС с различными узлами и элементами памяти. Одним из вариантов такого сочетания является комбинирующий генератор.

**Вариант РСЛОС в лабораторной работе:**

| **Вариант** | **РСЛОС №1** | **РСЛОС №2** | **РСЛОС №3** |
| --- | --- | --- | --- |
| **18** | **00110**  **01111** | **0011011**  **1011011** | **11001101**  **10010111** |

**Код:**

**import numpy as np**

**from copy import deepcopy**

**from typing import Tuple**

**class RSLOS:**

**def \_\_init\_\_(self, a: np.ndarray, c: np.ndarray):**

**assert a.shape == c.shape, f'Ожидали, что a и c будут иметь одинаковую форму, но получили: а: {a.shape}, c: {c.shape}'**

**self.a = a**

**self.c = c**

**self.b = deepcopy(a)**

**def external\_output(self) -> int:**

**return self.b[0]**

**def change\_internal\_state(self) -> None:**

**r = (self.c \* self.b).sum() % 2**

**for i in range(self.b.shape[0] - 1):**

**self.b[i] = self.b[i + 1]**

**self.b[-1] = r**

**def reset\_state(self) -> None:**

**self.b = deepcopy(self.a)**

**def generate\_sample(self, len\_: int) -> np.ndarray:**

**sample = np.zeros((len\_,))**

**for i in range(len\_):**

**sample[i] = self.external\_output()**

**self.change\_internal\_state()**

**return sample**

**class GepheGenerator:**

**def \_\_init\_\_(self, rslos\_1: RSLOS, rslos\_2: RSLOS, rslos\_3: RSLOS):**

**self.rslos\_1 = rslos\_1**

**self.rslos\_2 = rslos\_2**

**self.rslos\_3 = rslos\_3**

**def generate\_sample(self, len\_: int) -> np.ndarray:**

**x1 = self.rslos\_1.generate\_sample(len\_)**

**x2 = self.rslos\_2.generate\_sample(len\_)**

**x3 = self.rslos\_3.generate\_sample(len\_)**

**return np.logical\_xor((x1 \* x2), ((np.logical\_xor(x1, 1)) \* x3))**

**def reset\_state(self) -> None:**

**self.rslos\_1.reset\_state()**

**self.rslos\_2.reset\_state()**

**self.rslos\_3.reset\_state()**

**def count\_0\_and\_1(sample: np.ndarray) -> Tuple[int, int]:**

**ones = sample.sum()**

**zeros = sample.size - ones**

**return zeros, ones**

**def r\_i(sample: np.ndarray, i):**

**res = 0**

**for j in range(sample.size - i):**

**res += (-1) \*\* (sample[j] ^ sample[j + i])**

**return res**

**a\_1 = np.array([int(i) for i in "00110"])**

**c\_1 = np.array([int(i) for i in "01111"])**

**a\_2 = np.array([int(i) for i in "0011011"])**

**c\_2 = np.array([int(i) for i in "1011011"])**

**a\_3 = np.array([int(i) for i in "11001101"])**

**c\_3 = np.array([int(i) for i in "10010111"])**

**rslos\_1 = RSLOS(a\_1, c\_1)**

**rslos\_2 = RSLOS(a\_2, c\_2)**

**rslos\_3 = RSLOS(a\_3, c\_3)**

**gen = GepheGenerator(rslos\_1, rslos\_2, rslos\_3)**

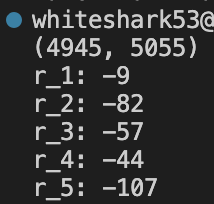
**gephe\_sample = gen.generate\_sample(10000)**

**print(count\_0\_and\_1(gephe\_sample))**

**for i in range(1, 6):**

**print(f'r\_{i}: {r\_i(gephe\_sample, i)}')**

**Результат:**

****