НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ” ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Лабораторна робота 2

«Методи реалізації криптографічних механізмів»

Виконали

Таран Вікторія ФБ-11мн

Рейценштейн Кирило ФБ-11мн

Київ 2022

Тема: “Реалізація алгоритмів генерації ключів гібридних криптосистем”.
Мета роботи: Дослідження алгоритмів генерації псевдовипадкових послідовностей, тестування простоти чисел та генерації простих чисел з точки зору їх ефективності за часом та можливості використання для гененерації ключів асиметричних криптосистем.

Завдання на лабораторну роботу
Для другого типу лабораторних робіт – аналіз стійкості реалізацій ПВЧ та генераторів ключів для обраної бібліотеки.
Варіанти завдань другого типу.
Підгрупа 2А. Бібліотека OpenSSL під Windows платформу.

Оформлення результатів роботи. Опис функції генерації ПСП та ключів бібліотеки з описом алгоритму, вхідних та вихідних даних, кодів повернення. Контрольний приклад роботи з функціями.

OpenSSL — це бібліотека програмного забезпечення для програм, які забезпечують безпечний зв’язок через комп’ютерні мережі від прослуховування або потребують ідентифікації сторони на іншому кінці. Основна бібліотека, написана на мові програмування C, реалізує основні криптографічні функції та надає різноманітні службові функції. Доступні обгортки, які дозволяють використовувати бібліотеку OpenSSL різними комп’ютерними мовами.

Ми обрали NIST Test Suite для перевірки ПВЧ. Там використані наступні тести:

- monobit
- frequency_within_block
- runs
- longest_run_ones_in_a_block
- dft
- non_overlapping_template_matching
- serial
- approximate_entropy
- cumulative sums
- random_excursion
- random_excursion_variant

Monobit. Тест фокусується на співвідношенні нулів і одиниць у всій послідовності. Мета цього тесту — визначити, чи кількість одиниць і нулів у послідовності є приблизно такою ж, як можна було б очікувати для справді випадкової послідовності. Тест оцінює близькість частки одиниць до ½, тобто кількість одиниць і нулів у послідовності має бути приблизно однаковою. Від проходження цього тесту залежать усі наступні тести.

Frequency_within_block. Тест фокусується на частці одиниць у M-бітних блоках. Метою цього тесту є визначення того, чи частота одиниць у M-бітовому блоці дорівнює приблизно M/2, як можна було б очікувати за припущення про випадковість. Для розміру блоку M=1 цей тест вироджується до тесту 1, тесту частоти (монобіт).

Runs. Цей тест зосереджується на загальній кількості циклів у послідовності, де цикл є безперервною послідовністю ідентичних бітів. Серія довжини k складається з рівно k однакових бітів і обмежена перед і після бітом протилежного значення. Мета перевірки прогонів полягає в тому, щоб визначити, чи кількість прогонів одиниць і нулів різної довжини відповідає очікуванням для випадкової послідовності. Зокрема, цей тест визначає, чи є коливання між такими нулями та одиницями занадто швидкими чи надто повільними.

Longest Run of Ones in a Block. Тест зосереджується на найдовшій серії одиниць у M-бітних блоках. Метою цього тесту є визначення того, чи відповідає довжина найдовшого ряду одиниць у перевіреній послідовності
довжина найдовшого циклу з тих, які можна було б очікувати у випадковій послідовності. Зауважте, що нерівномірність очікуваної довжини найдовшого ряду одиниць означає, що існує також нерівномірність очікуваної довжини найдовшого ряду нулів. Тому потрібен лише тест для одиниць.

Discrete Fourier Transform. Цей тест зосереджений на висотах піків у дискретному перетворенні Фур'є послідовності. Метою цього тесту є виявлення періодичних особливостей (тобто повторюваних візерунків, які знаходяться поруч) у досліджуваній послідовності, які б вказували на відхилення від припущення про випадковість. Мета полягає в тому, щоб виявити, чи кількість піків, що перевищують поріг у 95 %, значно відрізняється від 5 %.

Non-overlapping Template Matching. Цей тест зосереджується на кількості входжень попередньо визначених цільових рядків. Метою цього тесту є виявлення генераторів, які виробляють занадто багато випадків даного неперіодичного (аперіодичного) шаблону. Для цього тесту м-бітове вікно використовується для пошуку конкретного м-бітового шаблону. Якщо шаблон не знайдено, вікно пересувається на одну бітову позицію. Якщо шаблон знайдено, вікно скидається на біт після знайденого шаблону, і пошук відновлюється

Serial. Цей тест зосереджується на частоті всіх можливих накладених м-бітових шаблонів у всій послідовності. Мета цього тесту полягає в тому, щоб визначити, чи кількість повторень 2m м-бітових шаблонів, що перекриваються, є приблизно такою ж, як очікувалося б для випадкової послідовності. Випадкові послідовності мають рівномірність; тобто кожен m-бітовий шаблон має такий же шанс появи, як і будь-який інший m-бітовий шаблон.

Approximate Entropy. У центрі уваги цього тесту – частота всіх можливих збігів m-бітові шаблони по всій послідовності. Метою тесту є порівняння частоти накладання блоків двох послідовних/сусідніх довжин (m і m+1) із очікуваним результатом для випадкової послідовності.

Cumulative Sums. Цей тест фокусується на максимальному відхиленні (від нуля) випадкового блукання, яке визначається сукупною сумою скоригованих (-1, +1) цифр у послідовності. Мета тесту полягає в тому, щоб визначити, чи кумулятивна сума часткових послідовностей, що зустрічаються в тестованій послідовності, є занадто великою або занадто малою відносно очікуваної поведінки цієї кумулятивної суми для випадкових послідовностей. Цю кумулятивну суму можна розглядати як випадкове блукання. Для випадкової послідовності екскурсії випадкового блукання повинні бути

близькими до нуля. Для певних типів невипадкових послідовностей екскурсії цього випадкового блукання від нуля будуть великими.

Random Excursions. Цей тест зосереджується на кількості циклів, які мають рівно K відвідувань у кумулятивній сумі випадкового блукання. Кумулятивне сумарне випадкове блукання виходить із часткових сум після того, як послідовність (0,1) передається в відповідну послідовність (-1, +1). Цикл випадкового блукання складається з послідовності випадково взятих кроків одиничної довжини, які починаються з початку координат і повертаються до нього. Мета цього тесту полягає в тому, щоб визначити, чи кількість відвідувань певного стану в межах циклу відхиляється від того, що можна було б очікувати для випадкової послідовності. Цей тест насправді є серією з восьми тестів (і висновків), по одному тесту та висновку для кожного зі станів: -4, -3, -2, -1 і +1, +2, +3, +4.

Random Excursions Variant. Цей тест зосереджується на загальній кількості відвідувань певного стану (тобто зустрічається) у кумулятивній сумі випадкового блукання. Метою цього тесту є виявлення відхилень від очікуваної кількості відвідувань різних станів у випадковому блуканні. Цей тест насправді є серією з вісімнадцяти тестів (та висновків), по одному тесту та висновку для кожного зі станів: -9, -8, …, -1 та +1, +2, …, +9.

Код програми:

```
import cpuinfo
import numpy
import os
from nistrng import *

def get_rand_sequence():
    # returns binary array [ 0 1 0 1 .... 0 1 1 0 0 ]
    return pack_sequence(numpy.frombuffer(bytes.fromhex(os.popen("openssl rand -hex
200").read()), dtype=numpy.uint8))

def get_rand_sequence_numpy():
    return pack_sequence(numpy.random.randint(-255, 255, 200, dtype=int))

def main():
    print ("Used libs: cpuinfo, numpy, os, nistrng")
    print ("CPU where tests will be performed: " +
cpuinfo.get_cpu_info()['brand_raw'] + "\n")

    eligible_battery: dict = check_eligibility_all_battery(get_rand_sequence(),
SP800_22R1A_BATTERY)

    print("Eligible test from NIST-SP800-22r1a:")
    for name in eligible_battery.keys():
        print("-" + name)

    passed_data = dict()
    failed_data = dict()

    tests_num = 1000
```

```
    print ("Performing OpenSSL tests for " + str(tests_num) + " random binary
sequences...")

    for i in range (0, tests_num):
        binary_sequence = get_rand_sequence()

        print ("\nTesting given sequence: " + str(binary_sequence))

        results = run_all_battery(binary_sequence, eligible_battery, False)

        print("Test results:")
        for result, elapsed_time in results:
            if result.passed:
                data = passed_data.get(result.name)

                if data is None:
                    passed_data.update({ result.name: 1 })
                else:
                    passed_data.update({ result.name: data + 1 })
                print("- PASSED - score: " + str(numpy.round(result.score, 3)) + "
- " + result.name + " - elapsed time: " + str(elapsed_time) + " ms")
            else:
                data = failed_data.get(result.name)

                if data is None:
                    failed_data.update({ result.name: 1 })
                else:
                    failed_data.update({ result.name: data + 1 })
                print("- FAILED - score: " + str(numpy.round(result.score, 3)) + "
- " + result.name + " - elapsed time: " + str(elapsed_time) + " ms")

    print ("\nPassed tests amount: " + str(passed_data))
    print ("Failed tests amount: " + str(failed_data))

    passed_data.clear()
    failed_data.clear()

    print ("Performing NumPy tests for " + str(tests_num) + " random binary
sequences...")

    for i in range (0, tests_num):
        binary_sequence = get_rand_sequence_numpy()

        print ("\nTesting given sequence: " + str(binary_sequence))

        results = run_all_battery(binary_sequence, eligible_battery, False)

        print("Test results:")
        for result, elapsed_time in results:
            if result.passed:
```

```
                data = passed_data.get(result.name)

                if data is None:
                    passed_data.update({ result.name: 1 })
                else:
                    passed_data.update({ result.name: data + 1 })
                print("- PASSED - score: " + str(numpy.round(result.score, 3)) + "
- " + result.name + " - elapsed time: " + str(elapsed_time) + " ms")
            else:
                data = failed_data.get(result.name)

                if data is None:
                    failed_data.update({ result.name: 1 })
                else:
                    failed_data.update({ result.name: data + 1 })
                print("- FAILED - score: " + str(numpy.round(result.score, 3)) + "
- " + result.name + " - elapsed time: " + str(elapsed_time) + " ms")

    print ("\nPassed tests amount: " + str(passed_data))
    print ("Failed tests amount: " + str(failed_data))

if __name__ == '__main__':
    main()
```

Результати:
Ми згенерували 1000 чисел розміром 200 байт за допомогою openssl rand і numpy і перевірили їх з використанням NIST Test Suite.

```
Eligible test from NIST-SP800-22r1a:
-monobit
-frequency_within_block
-runs
-longest_run_ones_in_a_block
-dft
-non_overlapping_template_matching
-serial
-approximate_entropy
-cumulative sums
-random_excursion
-random_excursion_variant
Performing tests for 1 random binary sequences...

Testing given sequence: [0 1 0 ... 0 0 0]
Test results:
- PASSED - score: 0.764 - Monobit - elapsed time: 0 ms
- PASSED - score: 0.429 - Frequency Within Block - elapsed time: 1 ms
- PASSED - score: 0.121 - Runs - elapsed time: 0 ms
- PASSED - score: 0.081 - Longest Run Ones In A Block - elapsed time: 0 ms
- PASSED - score: 0.646 - Discrete Fourier Transform - elapsed time: 0 ms
- PASSED - score: 0.998 - Non Overlapping Template Matching - elapsed time: 1 ms
- PASSED - score: 0.67 - Serial - elapsed time: 29 ms
- PASSED - score: 0.154 - Approximate Entropy - elapsed time: 27 ms
- PASSED - score: 0.508 - Cumulative Sums - elapsed time: 1 ms
- PASSED - score: 0.384 - Random Excursion - elapsed time: 3 ms
- PASSED - score: 0.313 - Random Excursion Variant - elapsed time: 0 ms
```

Openssl:
Passed tests amount: {'Monobit': 985, 'Frequency Within Block': 991, 'Runs': 992, 'Longest Run Ones In A Block': 882, 'Discrete Fourier Transform': 988, 'Non Overlapping Template Matching': 984, 'Serial': 988, 'Approximate Entropy': 990, 'Cumulative Sums': 983, 'Random Excursion Variant': 632, 'Random Excursion': 28}
Failed tests amount: {'Random Excursion': 972, 'Longest Run Ones In A Block': 118, 'Random Excursion Variant': 368, 'Serial': 12, 'Monobit': 15, 'Cumulative Sums': 17, 'Non Overlapping Template Matching': 16, 'Discrete Fourier Transform': 12, 'Runs': 8, 'Approximate Entropy': 10, 'Frequency Within Block': 9}

Numpy:
Passed tests amount: {'Monobit': 989, 'Frequency Within Block': 996, 'Runs': 988, 'Longest Run Ones In A Block': 888, 'Discrete Fourier Transform': 982, 'Non Overlapping Template Matching': 995, 'Serial': 980, 'Approximate Entropy': 989, 'Cumulative Sums': 988, 'Random Excursion Variant': 673, 'Random Excursion': 15}
Failed tests amount: {'Random Excursion': 985, 'Random Excursion Variant': 327, 'Longest Run Ones In A Block': 112, 'Monobit': 11, 'Cumulative Sums': 12, 'Discrete Fourier Transform': 18, 'Non Overlapping Template Matching': 5, 'Serial': 20, 'Approximate Entropy': 11, 'Runs': 12, 'Frequency Within Block': 4}

Висновок:
98% з ПВЧ сгенерованих за допомогою openssl пройшли тести 'Monobit', 'Frequency Within Block', 'Runs', 'Discrete Fourier Transform', 'Non Overlapping Template Matching', 'Serial', 'Approximate Entropy' та 'Cumulative Sums'.
88% - 'Longest Run Ones In A Block'.
63% - Random Excursion Variant'.
3% - 'Random Excursion'.

Порівнюючи numpy з openssl, можна побачити схожі результати. Однак 'Random Excursion Variant' тест має показник на 4% вище і 'Random Excursion' на 1-2% нижче у numpy.

Як ми бачимо, результати з Random Excursion та Random Excursion Variant тестами є неуспішними. Це може свідчити про те, що в ПВЧ послідовність зустрічаються однакові стани. Загалом, більшість тестів була пройдена успішно.