Quiz 5

110550108 施柏江

Problem 1

a) Write a Python / C++ program to generate 1M bytes of cryptographically secure random numbers.

Ans:

I use secrets.token_bytes() to generate a secure random byte string of length 1024 * 1024.

```python
import secrets

n_bytes = 1024 * 1024
random_bytes = secrets.token_bytes(n_bytes)

with open("random.bin", "wb") as f:
    f.write(random_bytes)

print("Random numbers generated and saved to random.bin")
```

```
root@Brian-laptop:/mnt/c/Users/brian/OneDrive/桌面/hw/三下/密碼/hw5# python RNG.py
Random numbers generated and saved to random.bin
```

How to run: python RNG.py

b) 請簡單解釋各個檢測項目(Frequency, BlockFrequency, CumulativeSums...)的過程。

Ans:

1. Frequency: Checks the frequency of occurrence of 0s and 1s.

2. Block Frequency: Divides the data into blocks and examines the proportion of 0s and 1s within each block.

3. Cumulative Sums: Calculates partial sums.

4. Runs: Analyzes the number of consecutive identical bits.

5. Longest Run: Checks the length of the longest consecutive run of identical bits.

6. Rank: Assesses the distribution of matrix ranks.

7. FFT: Utilizes Fast Fourier Transform to examine spectral characteristics.

8. Non-Overlapping Template: Searches for the occurrence of specific templates within the data without allowing them to overlap.

9. Overlapping Template: Similar to the non-overlapping template test but allows templates to overlap.

10. Universal: Integrates multiple sub-tests.

11. Approximate Entropy: Measures the frequency of repetitive patterns.

12. Random Excursions: Counts the number of excursions away from zero.

13. Random Excursions Variant: A variation of the random excursions test that considers additional factors.

14. Serial: Assesses the correlation between different bits.

15. Linear Complexity: Evaluates the distribution of linear complexities.

```
root@Brian-laptop:/mnt/c/Users/brian/OneDrive/桌面/hw/三下/密碼/hw5# cat experiments/AlgorithmTesting/finalA
nalysisReport.txt
------------------------------------------------------------------------------
RESULTS FOR THE UNIFORMITY OF P-VALUES AND THE PROPORTION OF PASSING SEQUENCES
------------------------------------------------------------------------------
   generator is <random.bin>
------------------------------------------------------------------------------
 C1  C2  C3  C4  C5  C6  C7  C8  C9 C10  P-VALUE  PROPORTION  STATISTICAL TEST
------------------------------------------------------------------------------
  1   0   0   0   0   0   0   0   0   0     ----       1/1       Frequency
  1   0   0   0   0   0   0   0   0   0     ----       1/1       BlockFrequency
  1   0   0   0   0   0   0   0   0   0     ----       1/1       CumulativeSums
  1   0   0   0   0   0   0   0   0   0     ----       1/1       CumulativeSums
  0   0   0   0   0   0   0   1   0   0     ----       1/1       Runs
  0   0   0   1   0   0   0   0   0   0     ----       1/1       LongestRun
  0   0   0   0   0   0   0   1   0   0     ----       1/1       Rank
  0   0   1   0   0   0   0   0   0   0     ----       1/1       FFT
  1   0   0   0   0   0   0   0   0   0     ----       1/1       NonOverlappingTemplate
  0   0   0   1   0   0   0   0   0   0     ----       1/1       NonOverlappingTemplate
  1   0   0   0   0   0   0   0   0   0     ----       1/1       NonOverlappingTemplate
```

...

```
  0   0   1   0   0   0   0   0   0   0     ----       1/1       RandomExcursionsVariant
  0   0   1   0   0   0   0   0   0   0     ----       1/1       RandomExcursionsVariant
  0   0   1   0   0   0   0   0   0   0     ----       1/1       RandomExcursionsVariant
  0   0   0   0   0   0   0   0   0   1     ----       1/1       Serial
  0   0   0   0   0   0   0   0   0   1     ----       1/1       Serial
  0   0   0   0   1   0   0   0   0   0     ----       1/1       LinearComplexity


- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
The minimum pass rate for each statistical test with the exception of the
random excursion (variant) test is approximately = 0 for a
sample size = 1 binary sequences.

The minimum pass rate for the random excursion (variant) test
is approximately = 0 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.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

c) Extra credit: Find out a non-cryptographically secure random number generator, such as random(), to demonstrate its lack of safety. Then, propose modifications to enhance its security to generate cryptographically secure random numbers that meet the highest standards of security and reliability.

Ans:

```python
import random

n_bytes = 1024 * 1024
random_bytes = bytes(random.randint(0, 255) for _ in range(n_bytes))

with open("bonus/bonus.bin", "wb") as f:
    f.write(random_bytes)

print("Random numbers generated and saved to bonus/bonus.bin")
```

```
root@Brian-laptop:/mnt/c/Users/brian/OneDrive/桌面/hw/三下/密碼/hw5# python bonus/bonus.py
Random numbers generated and saved to bonus/bonus.bin
```

How to run: python bonus/bonus.py

(Can’t pass the test)

```
1   0   0   0   0   0   0   0   0   0     ----      0/1       NonOverlappingTemplate
0   0   0   0   1   0   0   0   0   0     ----      1/1       NonOverlappingTemplate
0   0   0   0   0   0   1   0   0   0     ----      1/1       NonOverlappingTemplate
0   0   0   0   0   0   0   0   1   0     ----      1/1       NonOverlappingTemplate
1   0   0   0   0   0   0   0   0   0     ----      0/1       NonOverlappingTemplate
```

The random.randint() function I used generates pseudo-random numbers using a pseudo-random number generator algorithm. This algorithm typically relies on an initial seed value to start generating numbers. If we know the seed value, we can predict the entire sequence of "random" numbers it generates. Moreover, the pseudo-random number generator has a finite period, after which it repeats its sequence of numbers. This characteristic makes it unsuitable for cryptographic applications where unpredictability and high entropy are crucial."

To enhance the security of random number generation, we can use the secrets module. Replace random.randint() with function such as secrets.token_bytes() just like I did in (a). This function is specifically designed for cryptographic purposes and ensures high security and reliability. Moreover, we can ensure that the random number generation process gathers sufficient entropy from the system. This ensures that the generated numbers are as unpredictable as possible.