Participants:

- **Pranav Deo – 104145**
- **Soroush Mostofi Rad – 107361**
- **Mohammadreza Mohebbi Najmabad – 106732**
- **Aurika Nurt – 106666**
- **Saeed Doozandeh – 107292**

Task 1.

Postprocessing of True Random Number Generators - Theoretical Background

1. In which cases is post-processing required for true random number generators?

TRNG usually use some natural physical phenomenon in order to extract the entropy and be the source of true randomness [such as race conditions, air turbulence in HDDs,mouse movement or any other noise], that is to make the system non-deterministic.

- Nevertheless, however random may the natural phenomenon source may seem, the digitalization of a physical phenomenon may introduce some form ofbias, usually some statistical bias.
- There might be implementation errors with the ICs or the ASICs (applicationspecific ICs) that are used to achieve the digitalization of the physical phenomenon, or the components could be defective (i.e., parasitic non ideal components).
- Sometimes sampling can introduce correlation.
- Thus, in order to resolve these issues, and covert any biased randomness (notreally random then, is it ?xD) bits to unbiased randomness, we require post processing techniques.
- Also, all physical random numbers seem to deviate from the statistical ideal. Post-processing is used to remove or reduce these deviations from the ideal.

2. Name 3 well known post processing techniques? What are their potential shortcomings?

A. Von-Neumann Correction Method

- Take two random bits and compare, if they are the same discard the bits, and if not then use the first bit.
- Why?

  Suppose that the Pb(0)==x && Pb(1)==y such that x!=y , then the Pb(01)==Pb(10)==x*y . The Pb(00)==x^2 || Pb(11)==y^2 are discarded.

B. Parity Based Post-Processing

- Break input stream into chunks each of a `n` bits.

- Compute a parity bit for each chunk by performing an XOR operation of all the bits in stream, discard the chunk and then use the parity bit.
- Why?

  Because for every `n` bit chunk taken, there happen to be exactly 2^(n-1) zero values and 2^(n-1) one values. As such performing the XOR operation increases chaos (entropy) leading to randomness.

- Problems: reduces the stream to 1/n, requires heavy computation for the same.

C. Universal Hashing

- Use hashing techniques (SHA 1, MD5) to convert arbitrary length random bits into a fixed length stream.

  Problem: Computationally heavy, more the bits, more heavy.

Task 2.

Postprocessing of True Random Number Generators - Theoretical Background

Output TRNG 1

```
00100000000001111110000000001001000
01111111111111100000100011010101
10000001110111111111111111100100
00000011111111001011101111111111
111110111110111110111010111101111
```

Output TRNG 2

```
00110110100110110010000111000010
00111110001101111101111000100001
01001000110100111110000111001100
001110000000001100010110001101000
11010110101010010010101010111001
```

1. **Compare the quality of the output of the two given TRNGs? What can you observe??**

https://medium.com/unitychain/provable-randomness-how-to-test-rngs- 55ac6726c5a3

<u>Note: How do we even really comprehend if something is random?</u>

Is `11111` random or is `0101010010101000101` random?

We can't say really by observation only!

Popular tests to check randomness:

- NIST RNG
- DieHarder: A Random Number Test Suite
- Knuth test

To test the randomness of a sequence, we first start by analyzing the source of entropy, and then we go after the 'deterministic' algorithm that uses the entropy seed and expands it into a sequence of keys.

Using Block Frequency testing:

- Considering the output of TRNG 1 as a block of streams, we will find the probability of 1 and 0 in each sub-block. We will then evaluate the mean ratios and see what the probability of each bit is.
- We are looking for the Pb (1) and Pb (0) to be as close to 0.5 as much as possible. When both the probabilities are 0.5, we can say that there is true randomness as the chance of both occurrences are equal.
- Should the probability skew, then we can say that this one.

Considering the output of TRNG 1 block of stream

```
[Sub-block 1]   00100000000011111100000000100100 ---> Pb(1)= 9/32 = 0.281 &&
Pb(0) == 23/32 = 0.718

[Sub-block 2]   01111111111111100000100011010101   ---> Pb(1)= 20/32 = 0.625
&& Pb(0) == 12/32 = 0.375

[Sub-block 3]   10000001110111111111111111100100   ---> Pb(1)= 21/32 = 0.656
&& Pb(0) == 11/32 = 0.343

[Sub-block 4]   00000011111111001011101111111111   ---> Pb(1)= 22/32 = 0.687
&& Pb(0) == 10/32 = 0.312

[Sub-block 5]   11111011111011110111010111101111   ---> Pb(1)= 26/32 = 0.812
&& Pb(0) == 6/32 = 0.187

Thus the total average Pb(1) = .6122
Thus the total average Pb(0) = .3878
```

Considering the output of TRNG 2 block of stream

```
[Sub-block 1]   00110110100110110010000111000010 ---> Pb(1)=  14/32 = 0.4375
&& Pb(0) == 18/32 = 0.5625

[Sub-block 2]   00111110001101111101111000100001 ---> Pb(1)=  14/32 = 0.4375
&& Pb(0) == 18/32 = 0.5625

[Sub-block 3]   01001000110100111110000111001100 ---> Pb(1)=  17/32 = 0.53125
&& Pb(0) == 15/32 = 0.46875

[Sub-block 4]   00111000000001100010110001101000 ---> Pb(1)=  10/32 = 0.3125
&& Pb(0) == 22/32 = 0.6875

[Sub-block 5]   11010110101010010010101010111001 ---> Pb(1)=  17/32 = 0.53125
&& Pb(0) == 15/32 = 0.46875

Thus the total average Pb(1) = .45
Thus the total average Pb(0) = .55
```

Observations:

- The sequence of TRNG 2 seems to be more random than TRNG 1 since the Pb (1) and Pb (0) are closer to being 0.5, thus there is less disparity in the frequency occurrence of 0 and 1.
- Even with quick glance at the sequence, the transitions of `1 -> 0` or `0 -> 1` (aka 'runs'), are less for sequence of TRNG 1. Less number of runs indicates poor randomness.
- Less number of runs in TRNG 1 also indicates bigger longest-run-of-m bits, i.e., the repletion of big blocks of either 1 or 0 are more in the TRNG1 sequence.

  (Here we don't know if these large blocks of a certain bit are truly random or not since we don't really know the source of entropy that resulted in the sequence).
- TRNG 2 sequence appears to be more random and hence of better quality than TRNG 1 sequence.

2. Apply the Von-Neumann correction to the random sequences.

- The von Neumann correction is used to remove any bias from a pseudo-random bit stream.
- It takes two input bits and outputs a single bit, only when there is a transition from the first bit to second.
- That is if the two input bits are same, then they are discarded and if they are not, then first bit is selected as output.
- Ideally the two bits are selected randomly, but for the sake of simplicity

TRNG 1

input	00	10	00	00	00	00	11	11	11	00	00	00	00	10	01	00
output		1												1	0	

input	01	11	11	11	11	11	11	10	00	00	10	00	11	01	01	01
output	1							1			1			0	0	0

input	10	00	00	01	11	01	11	11	11	11	11	11	11	10	01	00
output	1			0		0								1	0	

input	00	00	00	11	11	11	11	00	10	11	10	11	11	11	11	11
output									1		1					

input	11	11	10	11	11	10	11	11	01	11	01	01	11	10	11	11
output			1			1			0		0	0		1		

TRNG 2

input	00	11	01	10	10	01	10	11	00	10	00	01	11	00	00	10
output			0	1	1	0	1			1		0				1

input	00	11	11	10	00	11	01	11	11	01	11	10	00	10	00	01
output				1			0			0		1		1		0

input	01	00	10	00	11	01	00	11	11	10	00	01	11	00	11	00
output	0		1			0				1		0				

input	00	11	10	00	00	00	01	10	00	10	11	00	01	10	10	00
output			1				0	1		1			0	1	1	

input	11	01	01	10	10	10	10	01	00	10	10	10	10	11	10	01
output		0	0	1	1	1	1	0		1	1	1	1		1	0

3. Apply the Von-Neumann correction to the random sequences.

Method

- Divide stream in n bits.
- Calculate XOR calculation of the bits, and discard the chunk.

```
XOR TABLE REF

A    B   A XOR B

0    0      0
0    1      1
1    0      1
1    1      0
```

- Dividing the Sequence of TRNG 1 into chunk size of 16.

```
n=16

0010000000001111       1100000000100100    ---->    1      0
    (Chunk 1)               (Chunk 2)

0111111111111110       0000100011010101    ---->    0      0

1000000111011111       1111111111100100    ---->    1      0

0000001111111100       1011101111111111    ---->    0      0

1111101111101111       0111010111101111    ---->    0      0

Thus the sequence is reduced to:

00100000000011111100000000100100           10
01111111111111100000100011010101           00
10000001110111111111111111100100    -->    01
00000011111111001011101111111111           00
11111011111011110111010111101111           00
```

- Dividing the Sequence of TRNG 2 into chunk size of 16.

```
0011011010011011       0010000111000010    ---->    1     1

0011111000110111       1101111000100001    ---->    0     0

0100100011010011       1110000111001100    ---->    1     0

1101011010101001       0010101010111001    ---->    1     0

Thus the sequence is reduced to:

 00110110100110110010000111000010           11
 00111110001101111101111000100001           00
 01001000110100111110000111001100    -->    10
 00111000000001100010110001101000           10
 11010110101010010010101010111001           10
```

4. What do you observe when comparing these two post-processing methods?

- In the von Neumann correction, the TRNG1 sequence was reduced by 0.1375 [22/160] and TRNG sequence was reduced by 0.2437 [39/160].
- This sheds light on our earlier observation that TRNG 1 contains more sequence of constant m-bit blocks such as '111' or '0000'.
- Thus, the output of von Neumann method depends on the quality of the input sequence, and while truly random sequences will be reduced by 0.25, others will be greatly reduced.

Note: why does this happen?

[https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_520]

In the parity-based correction method, the sequence was always reduced by 1/n, i.e., 1/16 in our example.

Task 3.

Cellular automata shift register.

Assume the binary integers $10101111_2 = 175_{10}$ and $11110010_2 = 242_{10}$. Use these integers to produce the relevant cellular automata shift register, based on the following tables. Please fill in all the tables, including the two auxiliary ones.

(175) base_10 = (10101111) base_2
(242) base_10 = (11110010) base_2

Table 1: Cellular Automata Shift Register

Rule List	175	242	175	175	242	175	242	242
State 0	0	1	1	1	0	1	1	0
State 1	1	0	1	1	1	1	1	1

Rule Table for (175) base_10

Number	Neighborhood	Rule Set
7	1 11	1
6	1 10	0
5	1 01	1
4	1 00	0
3	0 11	1
2	0 10	1
1	0 01	1
0	0 00	1

Rule Table for (242) base_10

Number	Neighborhood	Rule Set
7	1 11	1
6	1 10	1
5	1 01	1
4	1 00	1
3	0 11	0
2	0 10	0
1	0 01	1
0	0 00	0

Task 4.
Physical Unclonable Functions - Theoretical Background

1. Three PUF application and how PUFs are used for it?

PUFs can be used for

- Identification and Authentication
- Storing keys and hashes
- Random Number Generators

2. Explain How optical PUFs work?

Optical PUFs work by scattering light over a transparent material that has some randomly scattered opaque spots/ particles scattered all over.

The opaque particle essentially blocks the light waves, and the result is a unique pattern (aka speckle pattern) that can be obtained on the other side of the transparent material.
This pattern is can be recorded to form a response database and can be post-processed as well.
Optical PUFs have quite a few issues:

- They require the availability of optical devices and optical readers.
- The optical measurements taken should be quite precise in order to create a proper response challenge database and later identify the challenge.

3. Think of a way of creating an optical PUF yourself with simple means?

https://www.degruyter.com/document/doi/10.1515/nanoph-2020-0049/html

In this paper, they introduce and demonstrate a robust optical PUF constructed from silicon photonic circuitry which can readily be interrogated from industry-standard wafer-scale fiber-optic probing and yields random, highly visible, and unclonable signatures with distinct features that are immune to probing and environmental variations. The robustness of our high-level approach is realized through the combination of several unique aspects. First, co-integration of a mode-filter and disordered photonic structure is employed to suppress the effect of probing variations. Secondly, we developed a photonic design that achieves very high sensitivity toward 'weak' perturbations

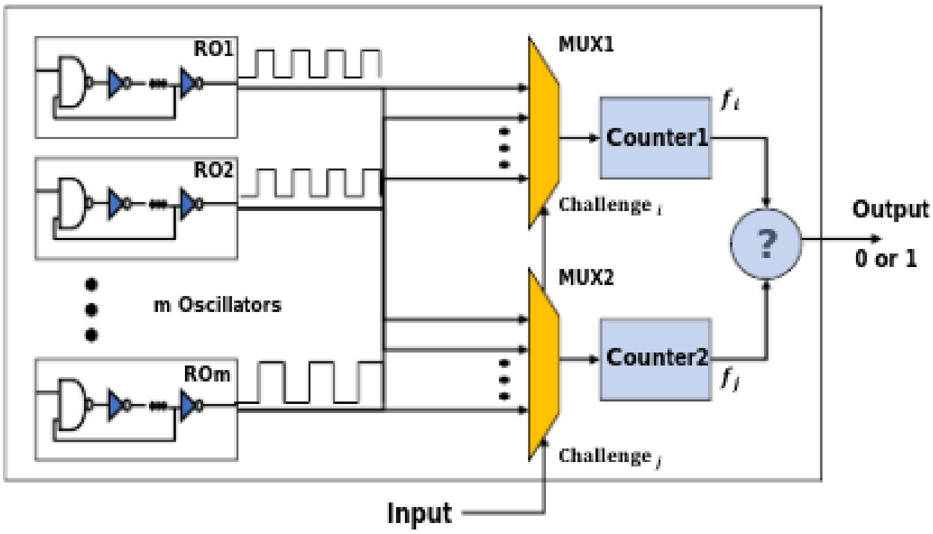
4. Explain how the ring oscillator PUF works? What it it's problem?

https://www.sciencedirect.com/topics/computer-science/ring-oscillator
https://www.researchgate.net/publication/355078122_Enhancing_the_Performance_of_Lightweight_Configurable_PUF_for_Robust_IoT_Hardware-Assisted_Security

A ring oscillator is a device that consists of an odd number of NOT gates laid in a ring fashion. The output of a ring oscillator varies between two voltage levels, each representing a binary 0 or 1 (true or false).

The reason that we have multiple numbers of NOT gates, and exactly an odd number of NOT gates, is that an odd number of odd gates means that the output is exactly the same as having one NOT gate, but the additional gates add a certain amount of delay.

This delay is random and cannot be controlled in any way (thus a good source of random entropy source).



- A ROPUF (Ring oscillator puf) relies on mapping m challenge bits to n response bits.
- The response bits ri (see image above) are usually determined by the manufacturer during the manufacturing process, as the fabrication method used usually leads to slight changes in the frequency of each ring oscillator.
- We have two challenge bits, $challenge_i$ and $challenge_j$, which are fed into the multiplexer.
- These challenge bits decide which ring oscillator will be selected by the multiplexers.
- The relative frequencies of the selected ring oscillators, say fi and fj, are then compared, by comparing the relative number of clock cycles over a given time span.
- The response bit is ri generated as follows:

```
ri = {1, if fi >= fj || 0, otherwise}
```

Problems with ROPUFS

- The problems with ROPUFs may lie in the ring oscillators itself.
- By design the ROs might not all work, and only some ROs might generate any response. This leads to a certain level dependency.
- Jitter is a common issue with ROs, such that deviation in temperatures can cause jitter, which may change the way an RO reacts to a challenge bit or change the frequency of RO causing undesired outputs.