Project Report
CS525 - Intro to Cryptography

Rahul Chowdary Garigipati
Long Tran

Project Topic

Our project was to implement 3 block cipher modes (CBC, OFB, and CTR) and compare the difference in their performance in terms of the total time taken to encrypt and decrypt a message.

Update Since Progress Report

Since the last project updates we have found out and learned the following main things:

- According to Ajitsaria, we cannot use threads in Python because it only exhibits concurrency and not parallelism due to Global Interpreter Lock (GIL).
- According to Python documentation, Python multiprocess does not work with methods of objects and objects that are not serializable:
    - This required us to modify our original implementation of OFB and more so CTR to allow for multiprocess. On the receiver's side of this new implementation, we will pre-calculate the numbers that are used to XOR with the cipher blocks, which will not be included in the time measurement. This will be further explained in the github repository and in Codes.
- AES implementation of PyCryptodome has a fixed block size of 16 bytes:
    - We modified the implementation of all 3 modes to divide a block into 16-byte blocks to encrypt/decrypt and reassemble them back together.

Code

Important points about the code:

- There is a sender and receiver, the sender will send the encrypted message to the receiver and the receiver will try to decode it. The protocol for sending is UDP.
- Only the decryption on the receiver's side will be done in parallel. The encryption on the sender's side will be done serially. However, for only OFB, there will be a pre-calculation of numbers that are used to XOR in the encryption, which is not time measured.
- The calculation of the numbers used to XOR with the cipher blocks on the receiver side of OFB and CTR will not be included in the time measurement.
  For OFB, it is because it can help us show the benefit of OFB being semi-parallelizable. However, for CTR, this is to be able to run CTR in a truly parallel fashion. In Python multiprocess, since it creates a new process, the function cannot contain pointers outside the function and can only take parameters that are serializable, so we have to pre-calculate the numbers and pass that to a function. Then, to show the benefit of parallelism, we did not include the time for pre-calculation.
  Yet, a normal version of the CTR is still available in its class definition.
- The block size for AES is fixed to be at 16, so to accommodate different block sizes, we modified the original implementation to divide the block into chunks of 16 bytes, encrypt/decrypt each chunk, and reassemble afterwards.

This also leads to a change in our program that will increment the block size to be a multiple of 16, if it is not so already.
- For OFB and CTR, each block will be decrypted by one process. For CBC, each block is handled by one thread, so decryption is done concurrently but not parallelly like OFB and CTR. This is to account for situations of random ordering of arrival blocks.

Github Link: https://github.com/ltran1612/CS525-CodingProject

Test Methodology

Test Machine

- CPU: AMD® Ryzen 7 5800H with 8 cores and 16 threads.
- RAM: 16 GB
- OS: 64-bit Ubuntu 22.04.1 LTS

Programming Language and Libraries

- Programming Language - Python 3.10.6 (CPython)
- The cryptographic library - PyCryptodome library for the encryption algorithm. Link: https://github.com/Legrandin/pycryptodome

Time Measurement

We used the perf_counter() of the Python "time" library to measure the time taken. This is a timer using a system-wide counter that allows measuring time elapsed over multiple separate processes. This is because it gives a system-wide timestamp with a shorter range but higher precision than using normal time. Link: https://docs.python.org/3/library/time.html#time.perf_counter

Test Metrics

Encryption/Decryption Algorithm

We tested with the following cipher algorithm:

- AES

Block Size/IV Size

We tested with the following block sizes:

- 16 bytes
- 500 bytes
- 1000 bytes
- 2000 bytes

Key Size

We tested with the following key size:

- 256 bits

Message Size

We will test with messages encoded in "utf-8" with the following size:

- 130 characters/bytes

- 20160 characters/bytes
- 48603 characters/bytes

Combinations Tested

We tested the following combinations:

- AES/16 bytes block size/130 characters message
- AES/1000 bytes block size/20160 characters message
- AES/500 bytes block size/48603 characters message
- AES/1000 bytes block size/48603 characters message
- AES/2000 bytes block size/48603 characters message

Data

CBC

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	61457973406400	6.15E+13	1.14E+06
Iteration 2	61459160318930	6.15E+13	1.02E+06
Iteration 3	61460377110422	6.15E+13	9.74E+05
Iteration 4	61461593778894	6.15E+13	1.10E+06
Iteration 5	61462788905153	6.15E+13	9.29E+05
Average	61460378703960	6.15E+13	1.03E+06

Table 1: Time taken by CBC on 130 characters message with block size of 16 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	59870604195744	5.99E+13	9.65E+06
Iteration 2	59874462535770	5.99E+13	9.70E+06
Iteration 3	59877917602957	5.99E+13	9.97E+06
Iteration 4	59880109500253	5.99E+13	9.85E+06
Iteration 5	59887433973158	5.99E+13	9.98E+06
Average	59878105561576	5.99E+13	9.83E+06

Table 2: Time taken by CBC on 20160 characters message with block size of 1000 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60492877544990	6.05E+13	2.14E+07
Iteration 2	60494467757739	6.05E+13	2.30E+07
Iteration 3	60496534464273	6.05E+13	2.06E+07
Iteration 4	60498405058133	6.05E+13	2.09E+07
Iteration 5	60501629636333	6.05E+13	2.14E+07
Average	60496782892294	6.05E+13	2.15E+07

Table 3: Time taken by CBC on 48603 characters message with block size of 1000 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60676429896023	6.07E+13	2.34E+07
Iteration 2	60683424268022	6.07E+13	2.32E+07
Iteration 3	60685347670852	6.07E+13	2.39E+07
Iteration 4	60687352250447	6.07E+13	2.30E+07
Iteration 5	60689392261095	6.07E+13	2.25E+07
Average	60684389269288	6.07E+13	2.32E+07

Table 4: Time taken by CBC on 48603 characters message with block size of 500 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	61591409066327	6.16E+13	2.09E+07
Iteration 2	61592569273052	6.16E+13	2.02E+07
Iteration 3	61593693372545	6.16E+13	1.99E+07
Iteration 4	61594841455871	6.16E+13	2.03E+07
Iteration 5	61595973049910	6.16E+13	2.02E+07
Average	61593697243541	6.16E+13	2.03E+07

Table 5: Time taken by CBC on 48603 characters message with block size of 2000 bytes in nanoseconds.

OFB

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	61323326095795	6.13E+13	1.27E+07
Iteration 2	61331232906207	6.13E+13	1.31E+07
Iteration 3	61332461325788	6.13E+13	1.32E+07
Iteration 4	61333537612843	6.13E+13	1.34E+07
Iteration 5	61334602005365	6.13E+13	1.40E+07
Average	61331031989200	6.13E+13	1.33E+07

Table 6: Time taken by OFB on 130 characters message with block size of 16 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60002467237493	6.00E+13	1.31E+07
Iteration 2	60005808092599	6.00E+13	1.23E+07
Iteration 3	60008077785427	6.00E+13	1.15E+07
Iteration 4	60010348369288	6.00E+13	1.15E+07
Iteration 5	60012500677387	6.00E+13	1.10E+07
Average	60007840432439	6.00E+13	1.19E+07

Table 7: Time taken by OFB on 20160 characters message with block size of 1000 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60398363362276	6.04E+13	1.41E+07
Iteration 2	60400544328970	6.04E+13	1.22E+07
Iteration 3	60404007530759	6.04E+13	1.21E+07
Iteration 4	60405616147156	6.04E+13	1.31E+07
Iteration 5	60407113337862	6.04E+13	9.30E+06

Average	60403128941405	6.04E+13	1.22E+07

Table 8: Time taken by OFB on 48603 characters message with block size of 1000 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60776374356093	6.08E+13	1.99E+07
Iteration 2	60779728107243	6.08E+13	1.90E+07
Iteration 3	60781735010877	6.08E+13	1.97E+07
Iteration 4	60783760130929	6.08E+13	1.94E+07
Iteration 5	60785647500545	6.08E+13	1.87E+07
Average	60781449021137	6.08E+13	1.93E+07

Table 9: Time taken by OFB on 48603 characters message with block size of 500 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	61196780016353	6.12E+13	8.55E+06
Iteration 2	61198340487266	6.12E+13	1.03E+07
Iteration 3	61199811955865	6.12E+13	9.79E+06
Iteration 4	61201174766712	6.12E+13	9.65E+06
Iteration 5	61202527959953	6.12E+13	9.99E+06
Average	61199727037230	6.12E+13	9.66E+06

Table 10: Time taken by OFB on 48603 characters message with block size of 2000 bytes in nanoseconds.

CTR

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	61507904646510	6.15E+13	1.25E+07
Iteration 2	61508978444343	6.15E+13	1.26E+07
Iteration 3	61510090581750	6.15E+13	1.31E+07

Iteration 4	61511165393150	6.15E+13	1.27E+07
Iteration 5	61512868998224	6.15E+13	1.27E+07
Average	61510201612795	6.15E+13	1.27E+07

Table 11: Time taken by CTR on 130 characters message with block size of 16 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60103998109019	6.01E+13	1.38E+07
Iteration 2	60106546323067	6.01E+13	1.36E+07
Iteration 3	60109269347408	6.01E+13	1.40E+07
Iteration 4	60111622382344	6.01E+13	1.44E+07
Iteration 5	60114054660053	6.01E+13	1.43E+07
Average	60109098164378	6.01E+13	1.40E+07

Table 12: Time taken by CTR on 20160 characters message with block size of 1000 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60271733469628	6.03E+13	1.84E+07
Iteration 2	60273661799502	6.03E+13	1.87E+07
Iteration 3	60275639896925	6.03E+13	1.87E+07
Iteration 4	60277644128737	6.03E+13	1.94E+07
Iteration 5	60279523761041	6.03E+13	1.93E+07
Average	60275640611167	6.03E+13	1.89E+07

Table 13: Time taken by CTR on 48603 characters message with block size of 1000 bytes in nanoseconds.

	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	60937522720552	6.09E+13	2.65E+07
Iteration 2	60939253856529	6.09E+13	2.29E+07

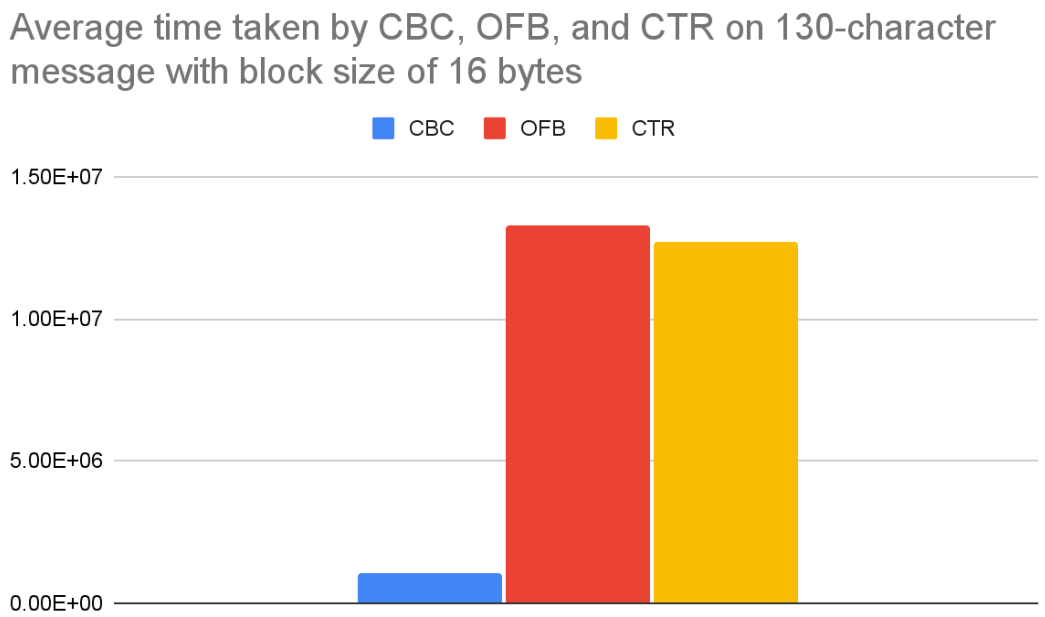
Iteration 3	60940940945133	6.09E+13	2.54E+07
Iteration 4	60942666395210	6.09E+13	2.52E+07
Iteration 5	60944121387538	6.09E+13	2.58E+07
Average	60940901060992	6.09E+13	2.52E+07

Table 14: Time taken by CTR on 48603 characters message with block size of 500 bytes in nanoseconds.

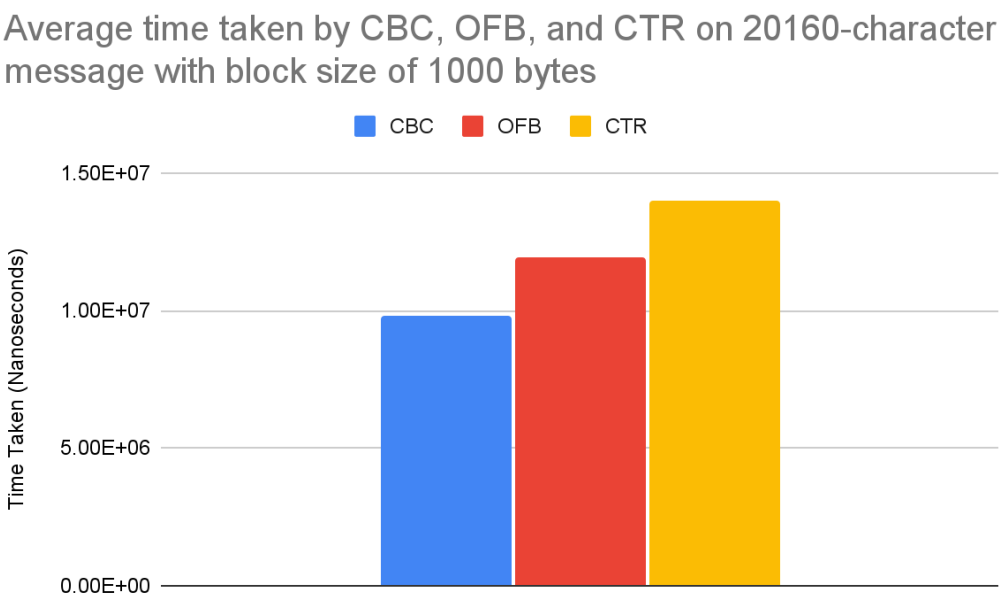
	Sender (Start Time in Nanoseconds)	Receiver (End Time in Nanoseconds)	Time Elapsed
Iteration 1	61004259773011	6.10E+13	1.69E+07
Iteration 2	61005706698192	6.10E+13	1.52E+07
Iteration 3	61006963935501	6.10E+13	1.63E+07
Iteration 4	61008143378740	6.10E+13	2.07E+07
Iteration 5	61009326312499	6.10E+13	1.50E+07
Average	61006880019589	6.10E+13	1.68E+07

Table 15: Time taken by CTR on 48603 characters message with block size of 2000 bytes in nanoseconds.

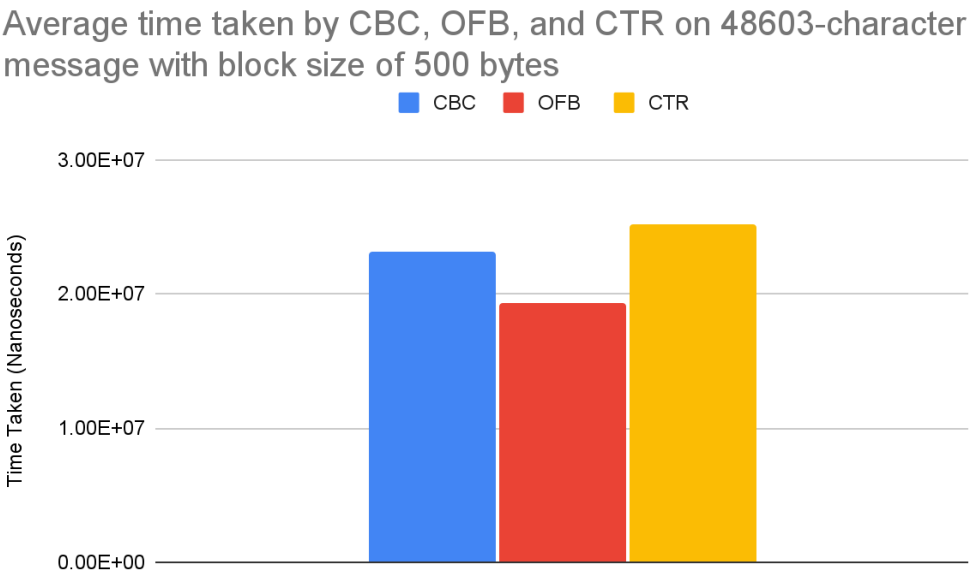
Graphs



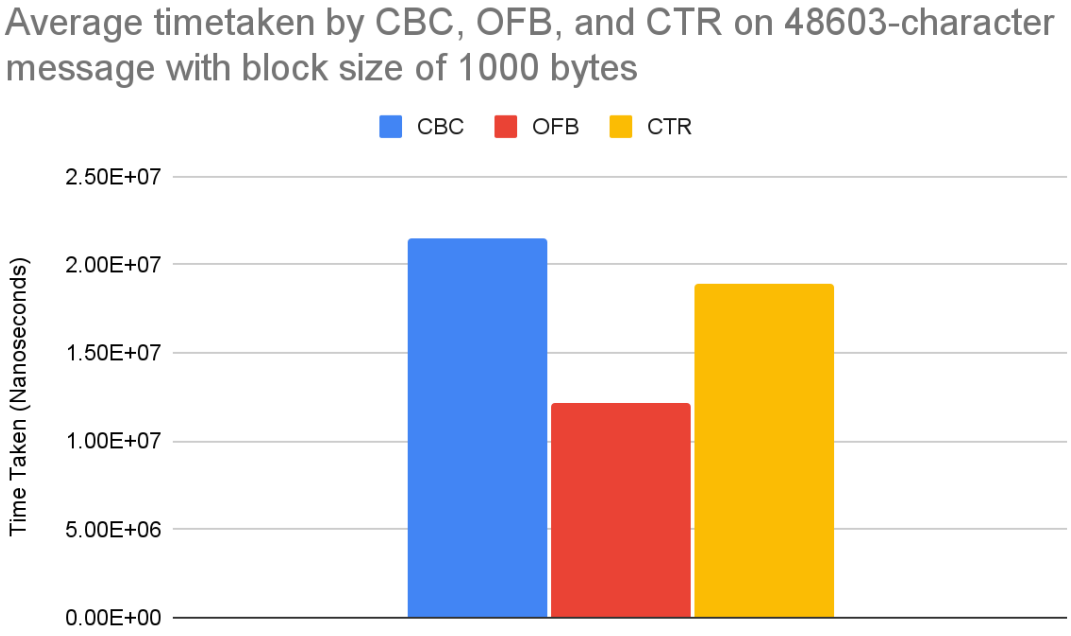
Graph 1



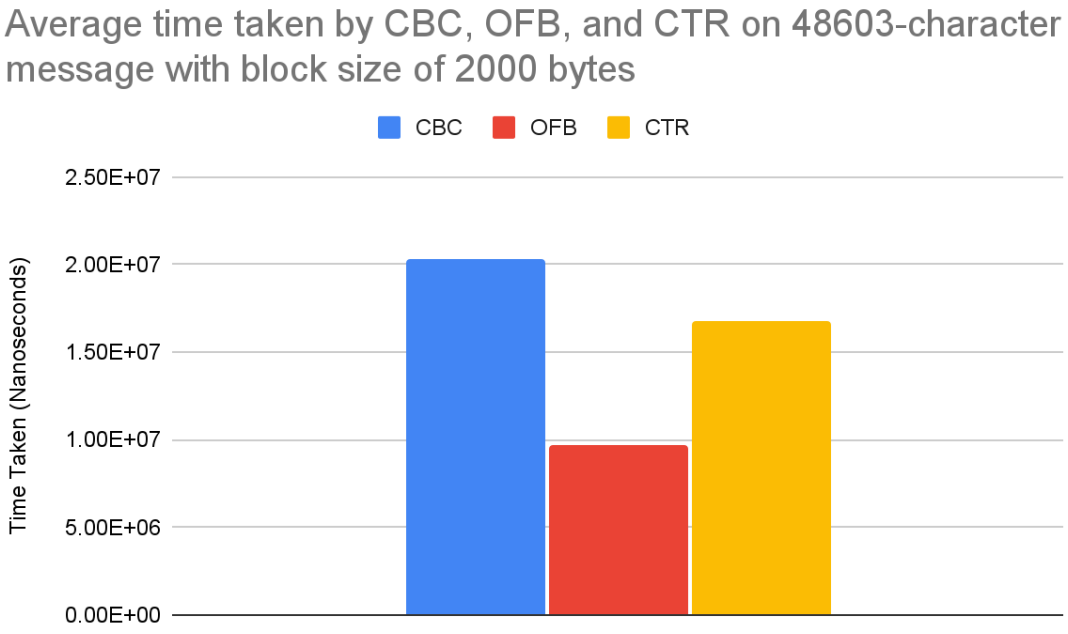
Graph 2



Graph 3



Graph 4



Graph 5

Analysis

We made the following observations from the graph.

1. From the above observation, we can say that the time taken for OFB of **block size 16 bytes** with the **message size 130- characters** is higher than CBC, and CTR, whereas CBC takes the least amount of time compared to OFB, and CTR, and CTR takes much higher time compare to CBC and less time compared to OFB.
2. For a **block size of 1000 bytes** with the **message size 20160 - characters**, CTR takes much higher time compared to CBC, and OFB, whereas CBC takes the least amount of time compared to OFB, and CTR. OFB takes slightly higher time compared to CBC and less time compared to CTR.
3. For a **block size of 500 bytes** with the **message size 48603 - characters,** CTR takes slightly higher time compared to CBC, and OFB, whereas CBC takes slightly higher time compared to OFB and less time compared to CTR. OFB takes the least amount of time compared to CBC, and CTR.
4. For a **block size of 1000 bytes** with the **message size 48603 - characters**, CBC takes much higher time compared to OFB, and CTR, whereas OFB takes the least amount of time compared to CBC, and CTR. CTR takes much higher time compared to OFB, and less time compared to CBC.
5. For a **block size of 2000 bytes** with the **message size 48603 - characters**, CBC takes much higher time compared to OFB, and CTR, whereas OFB takes the least amount of time compared to CBC, and CTR. CTR takes much higher time compared to OFB, and less time compared to CBC.

From observation 1 and 2, it showed that OFB and CTR are slower. This is probably because the ratio between the message size and block size is so small that there is less heavy work to do in each decryption compared to the work of creating and maintaining processes. Hence, the cost of creating/maintaining processes outweighs the benefits of parallelism, so CBC with its serial execution is faster.
From observation 3, it showed that OFB is faster than CBC, so the benefits from parallelism outweighs the cost of managing processes. However, CTR is slightly slower than CBC, which could be because on the sender's side, we did not do pre-calculation for CTR, so the benefits of parallelism on the receiver's side is not significant enough for CTR.
However, from observation 4 and 5, it clearly showed that both OFB and CTR are faster than CBC. This is because each process of OFB and CTR on the receiver's side has enough work to do that it outweighs the cost of managing them. OFB is faster than CTR because we do not have the pre-calculation for CTR on the sender's side as mentioned.

Conclusion

In conclusion, although our CTR implementation derives a little bit from the original design and our program is not the most efficient, we can see that OFB and CTR can give us some performance benefits from parallelism. However, just like any parallel program, there is a trade-off between the cost of creating threads/processes vs the time benefits gained from it. As a result, to make the most out of OFB and CTR, implementations need to choose the block size wisely relative to the message size so that the time benefits of parallelism outweigh its cost.

References

- Ajitsaria A. *What Is the Python Global Interpreter Lock (GIL)?*. RealPython. Retrieved from https://realpython.com/python-gil/
- Python Documentation. *multiprocessing — Process-based parallelism.* Retrieved from https://docs.python.org/3/library/multiprocessing.html