Implement a distributed consensus algorithm using PHE and OT

The goal of this exercise is to implement a distributed average consensus scheme.

You need Python for this assignment.

Some special modules are needed:

- *Paillier*
- *A module to help implement Oblivious Transfer (I used this OT module)*

Consider the following distributed consensus problem:

$$\min_{x_i \in \mathbb{R}} \frac{1}{2} \sum_{i=1}^{n} x_i^T q_i x_i$$

where $n = 3$, and $x_1^0 = 1,\ x_2^0 = 0.3, x_3^0 = 0.1,\ q_1 = q_2 = q_3 = 1.$

Rewrite the problem as follows:

$$\min_{x_i \in \mathbb{R}} \frac{1}{2} \sum_{i=1}^{n} x_i^T q_i x_i$$
$$\text{subject to } x_i = \bar{x}$$

You can consider the x_i as local variables and $\bar{x}$ as a global variable. You can solve the problem using the alternating direction method of multipliers (ADMM). ADMM iteratively solves the following problem ($\rho = 1,\ \text{iter\_max} = 18$):

```
for k = 0, 1, 2, …, iter_max
    for i = 1, 2, 3
        $x_i^{k+1} = \text{argmin}_{x_i}(x_i^T q_i x_i + \frac{\rho}{2} \parallel x_i - \bar{x}^k + u_i^k \parallel^2)$
    end
    $\bar{x}^{k+1} = \frac{1}{n}\sum_{i=1}^{n} x_i^{k+1}$
    for i = 1, 2, 3
        $u_i^{k+1} = u_i^k + x_i^{k+1} - \bar{x}^{k+1}$
    end
end
```

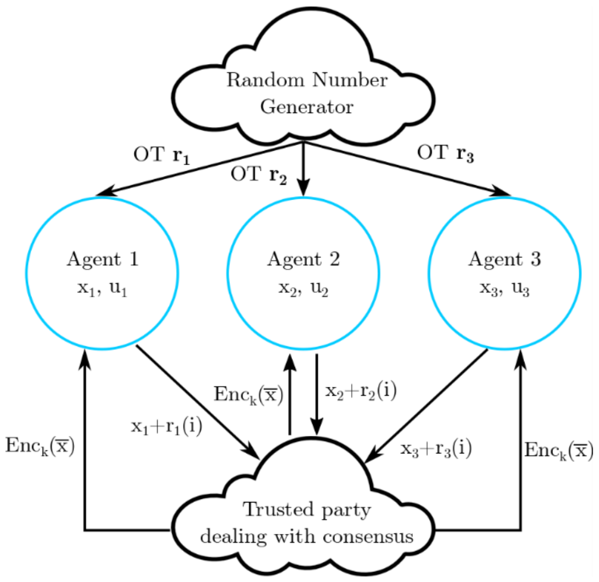
[1 pts] Solve the problem above in plaintext. Then,

- Report the values of each x_i and final $\bar{x}$ in one plot.
- Analyze the computation time of each iterate of the algorithm and the overall solving time.
- Attach the code to your submission with a README file.

[4 pts] Implement the ADMM-based consensus scheme described above by considering the following. Each agent does not trust the other agents. They require a trusted party to compute the sum $\bar{x}$. The trusted party receives over a secure network (assume that the network exists) the local updates x_i from the agents and returns the encrypted value of the sum to all the agents, namely $\mathrm{Enc}_k(\bar{x})$. The updates of x_i, u_i will be over encrypted values. Once the agents will reach consensus the trusted party will reveal their values.

- Can you obtain the same values x_i and $\bar{x}$ as in the original plaintext implementation? Plot the (decrypted) values of x_i at each iterate.
- Analyze the computation time of each iterate of the algorithm and the overall solving time. Do you see a difference compared to the previous case?
- Attach the code to your submission with a README file.

[5 pts] Implement the ADMM-based consensus scheme described in the figure. Compared to the previous point each agent does not trust the other agents and the "Trusted party" dealing with consensus. Hence, another party is introduced in the scheme. This party has the task of generating a vector of random numbers that the agents can use as a 1-time pad to hide their local values to the "Trusted party". Notice that the random numbers should be generated such that their introduction does not affect the computation of $\bar{x}$ and at the same time can obfuscate the real values of x_i. To exchange the values between the "Random Number Generator" and the agents an oblivious transfer protocol is required. Use one of the existing python modules for this.



- Verify that you obtain the same values x_i and $\bar{x}$ as in the original plaintext implementation. Plot the (decrypted) values of x_i at each iterate.
- Analyze the computation time of the oblivious transfer protocol and discuss the results.
- Attach the code to your submission with a README file.