Crypto Assignment

Mart Veldkamp

I. Introduction

In this assignment, we are tasked with implementing a distributed average consensus scheme.

In the first implementation, we implemented an ADMM-based consensus scheme where every agent could freely communicate with each other.

In the second implementation, the parties do not trust each other and require a trusted third party to perform the computation.

Finally, we consider a case where the parties do not trust either the third party or each other.

II. Deriving the convergence

As given in the assignment, the following formulas are used. The local update is:

$$x_i^{k+1} = \text{argmin}_{x_i}(x_i^T q_i x_i + \frac{\rho}{2}||x_i - \bar{x}^k + u_i^k||^2)$$

Where x_i is the local variable, and $\bar{x}$ is the global variable.

Next we have the Global (Consensus) Update:

$$\bar{x}^{k+1} = \frac{1}{n}\sum_{i=1}^{n} x_i^{k+1}$$

And the Dual Update:

$$u_i^{k+1} = u_i^k + x_i^{k+1} - \bar{x}^{k+1}$$

We can rewrite the local update into the following minimization form, give $\rho = q_i = 1$:

$$f(x_i) = x_i^2 + \frac{1}{2}(x_i - (\bar{x}^k - u_i^k))^2$$

Then, we take the derivative and set it to 0 (since we want to find the minimum), which gives:

$$2x_i + (x_i - (\bar{x}^k - u_i^k)) = 0$$

Finally, at convergence, we can rewrite x_i and $\bar{x}^k$ as x^*, and u_i^k as u^*, to:

$$2x^* + u^* = 0 \rightarrow x^* = -\frac{1}{2}u^*$$

because we set $u_0 = x_0$, and we know $x_0 = [1, 0.3, 0.1]$. We can calulcate the mean, which is $\frac{1+0.3+0.1}{3} \approx 0.466$, which we now can fill into the formula to get the convergence, which is: $x^* \approx -0.233$.

Mart Veldkamp is a master's students at TU Delft, The Netherlands. E-mail:mmveldkamp@student.tudelft.nl.

III. Trusted ADMM

We first consider the case where each agent trusts the others, and therefore makes it so that $x_i = \bar{x}$. After running the simulation, the following results were obtained:

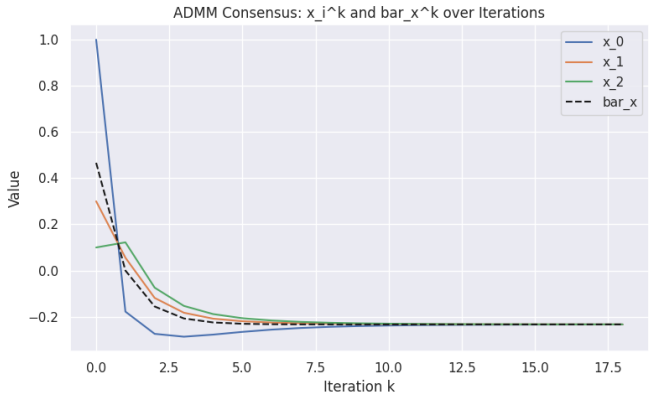

Fig. 1. All trusted ADMM Consensus.

Mean iteration time:	$\approx 2 \cdot 10^{-05}$ s
Overall solving time:	≈ 0.0005 s

TABLE I
Time table trusted ADMM

As seen in Figure 1, we observe clear convergence to the expected value, derived in the previous chapter. We can also see that the algorithm is very fast. Almost no visible delay between running the script and the output. This is expected, as we are only using basic arithmetic operations. However, this serves as a good baseline for how a 'no privacy' algorithm would perform.

IV. Paillier-based ADMM

Next, we consider the case where each agent does not trust the others. They require a trusted party to compute the sum $\bar{x}$.

In this implementation, each agent is assigned a key, with which they can send encrypted data to a shared trusted party. This third party will then compute the values and send back these values. As expected, these additional calculations introduce a longer runtime. As can be seen from II.

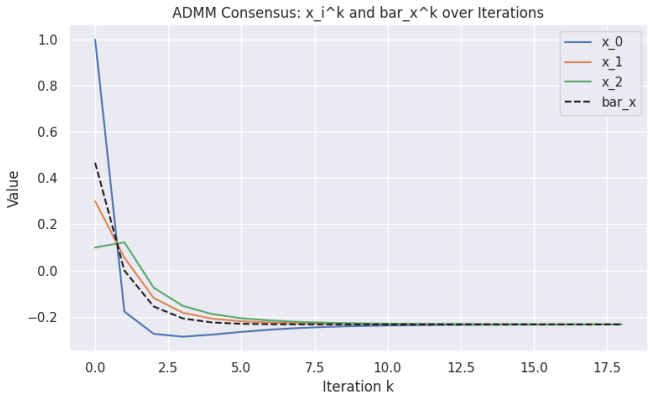

Fig. 2. All trusted ADMM Consensus.

Mean iteration time:	≈ 0.74 s
Overall solving time:	≈ 13.8 s

TABLE II
Time table Paillier

And as can be seen from 2, the consensus curve didn't change, this is expected as we only influence the communication between the agents, not the values itself. However, it is still significantly slower than before (by a factor of 27,600).

V. Oblivious Transfer -Based ADMM

Finally, we introduce Oblivious Transfer (OT). Where each party does not trust either the other parties or the entity computing the consensus. Because of this, we add a fourth party.

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".

To exchange values between the 'Random Number Generator' and the agents, an oblivious transfer protocol is required.

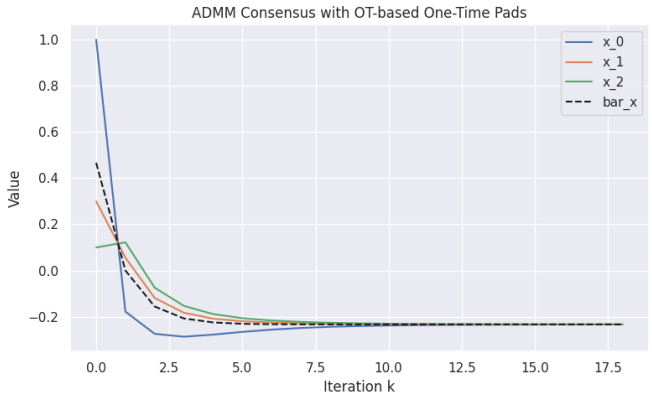

Fig. 3. ADMM Consensus with OT implementation.

Mean iteration time:	$\approx 6 \cdot 10^{-05}$ s
Overall solving time:	≈ 0.0012 s

TABLE III
Time table with OT implementation

As seen in Figure 3, the decrypted data still reaches consensus—and much faster. This is because, as mentioned earlier, we use the random vector only once to hide the initial values of each agent. After this we use only a slightly changed method as with the trusted ADMM.

This results in a fast and secure protocol but requires an additional party.