Decentralized Machine Learning Models with Cryptography Techniques

AMAN

Mentors:- Prof. Siba Narayan Swain,

Prof. Mahadeva Prasanna

BTP Presentation 2020

Problems With Centralized Approach?

There are mainly 3 major risks involved in the Centralized approach. These are the following risks mentioned below:-

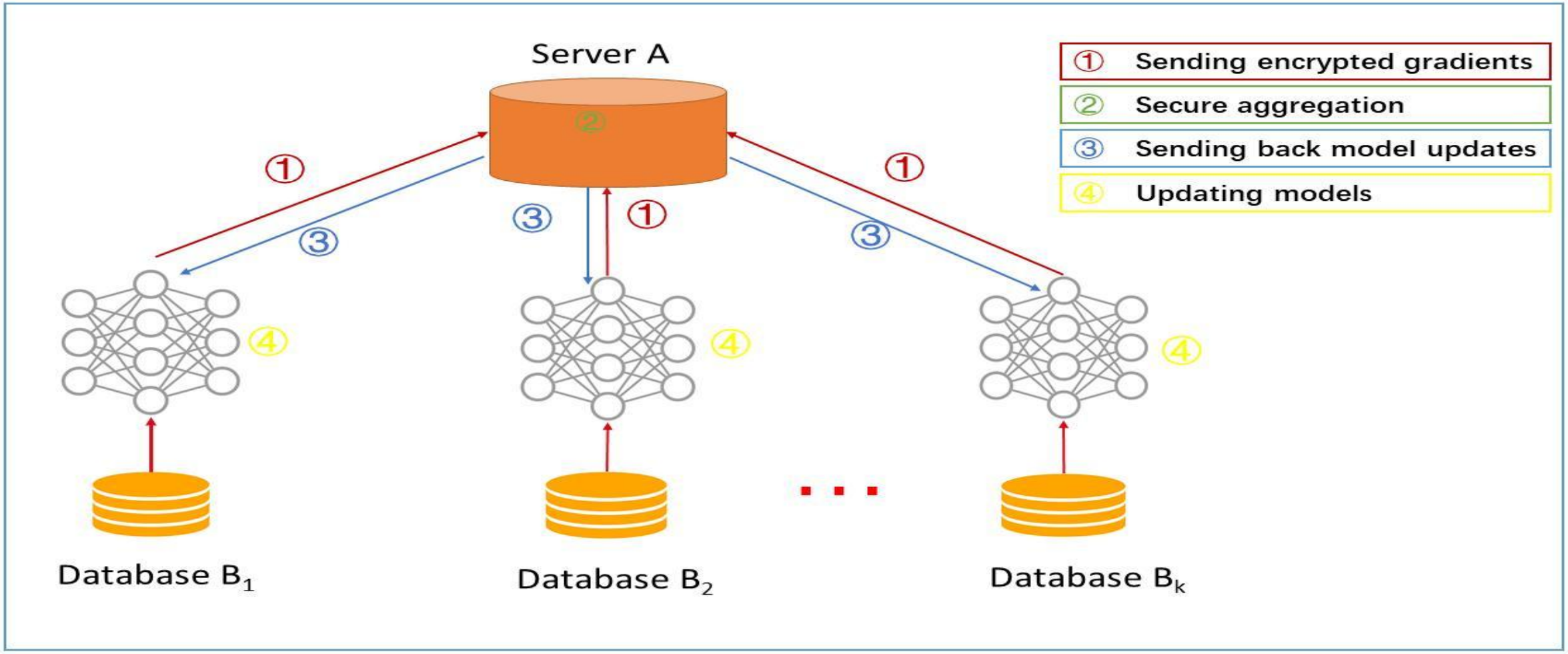
1. Rich Get Richer Problem:- Big Companies have advantages over the small companies or startups as they have the huge data to train a Model. Moreover, they can decide to run different model on this proprietary data**.**

2. Transparency Problem:- These Model are increasingly important in our lives. But we know very little about them? How was it trained? Or who trains it?

3. Privacy Issue:- In this modern era, privacy of users is of major concern because when the data which should be kept private gets in the wrong hands.

"The future is here, It's just not evenly distributed ~ William Gibson"

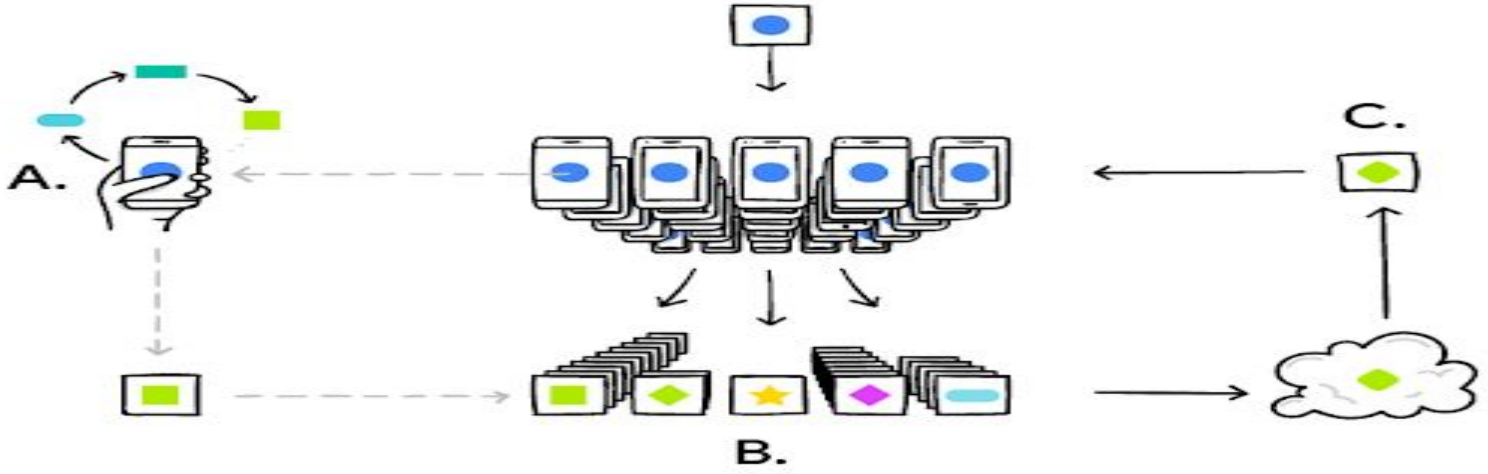
Solution:- Decentralized Machine Learning Models



"The future is here, It's just not evenly distributed ~ William Gibson"

Federated learning in real world

It is an approach that downloads the current model and computes an updated model at the device itself (a.k.a edge computing) using local data. These locally trained models are then sent from the devices back to the central server where they are aggregated, i.e. averaging weights, and then a single consolidated and improved global model is sent back to the devices



"The future is here, It's just not evenly distributed ~ William Gibson"

Maths Behind Federated Learning

1. Let there are **H1 , H2 , ..., Hn** are the hidden layers of the neural network, i.e. if **Wj,1** is a m*n matrix

$$W_{j,1} = W(I, H_1), W_{j,2} = W(H_1, H_2), \quad (1)$$
$$..., W_{j,i} = W(H_{i-1}, H_i),$$
$$..., W_{j,n+1} = W(H_n, O)$$

corresponding to layer 1 of client j where I and O are the input and Output layers.

2. We can represent Wj as a set of matrix containing weights of all the layers for a particular client j

$$W_j = \{W_{j,1}, W_{j,2}, ..., W_{j,n+1}\} \quad (2)$$

"The future is here, It's just not evenly distributed ~ William Gibson"

Continued...

3. Now at the server we aggregate each layers from different clients:-

$$W_{avg}^{t} = \frac{1}{K} \sum_{i=1}^{K} W_{i}^{t} \qquad (3)$$

where k are the no. of clients in a system and t is a time in seconds.

Challenges with the General FL Approach

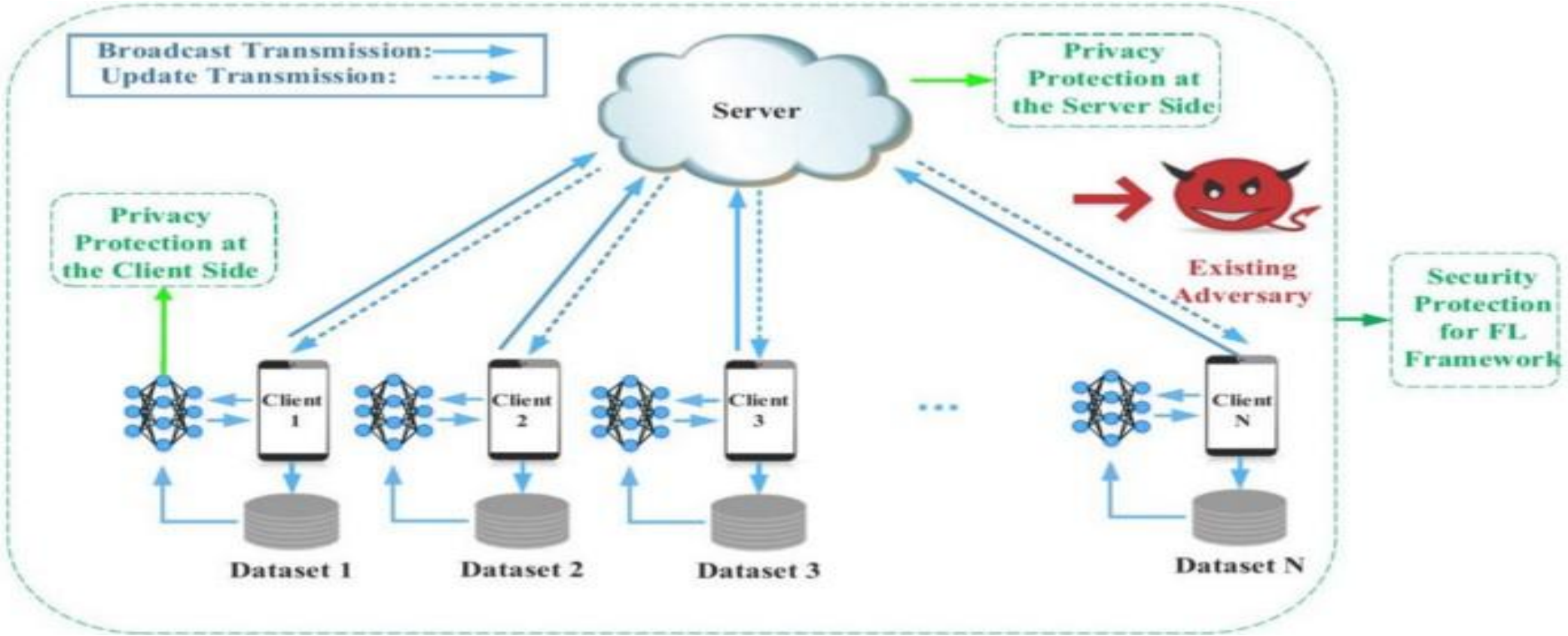
There are 2-3 major risks involved in the general federated learning approach.

1. Privacy Leak- An Issue Caused by insecure communication where the privacy of the shared parameters of the clients can be leaked and server can also learn about the shared weights or parameters.

Fig. 1: Reconstruction of an input image x from the gradient $\nabla_\theta \mathcal{L}_\theta(x, y)$ of a ResNet-18. Left: Image from the validation dataset. Middle: Reconstruction from an untrained ResNet-18. Right: Reconstruction from a ResNet-18 trained on ImageNet. In both cases, the intended privacy of the image is broken.

Continued...

2. Data Poisoning:- In FL, clients, who previously acted only as passive data providers, can now observe intermediate model states and might contribute arbitrary updates as part of the decentralized training process.



"The future is here, It's just not evenly distributed ~ William Gibson"

Continued...

3. Model Averaging:- The aggregation is mainly processed at the server after collecting individuals' parameters, and updates the global model. This process is particularly important as it should absorb the advantages of the clients and determine the end of learning. If protection method is applied at the client side, such as the perturbation applied before collecting model parameters, the aggregation cannot be simply a conventional averaging process.

Solutions:- Here Cryptography comes into the picture.

"Some secrets are meant to stay secret forever ~ Liane Moriarty"

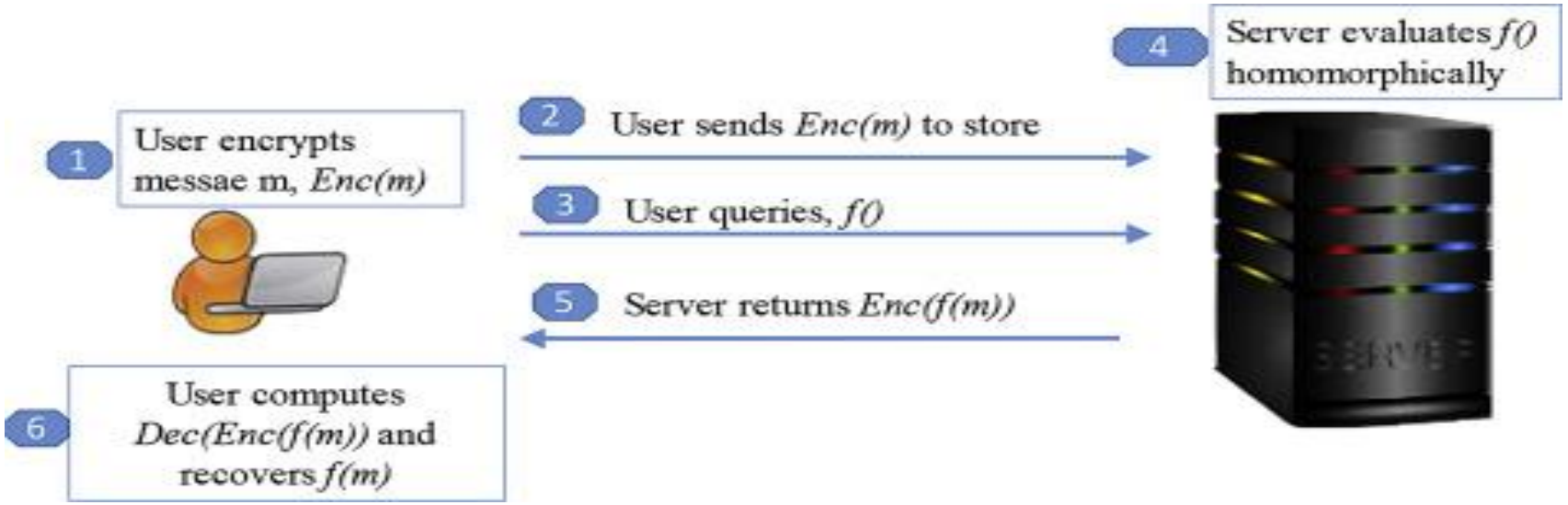
Homographic Encryption with Authentication

Homographic Encryption:-

Homographic Encryption is a public key cryptographic scheme. The user creates a pair of secret and public key, uses the public one to encrypt the data, before sending it to a third party which will perform computations on the encrypted data. Because of the homomorphic properties of the encryption and decryption, the user can get the encrypted result and decode it with her own key to see the output of the computation on their data, without having shown it once in clear to the third party. It admits some kind of computations on encrypted data.

The key point here is “publicly”, meaning that carrying out this computation has to be possible without having access to any secret information.

Homographic Encryption Continued..



Homomorphic encryption is adopted to protect user data through parameters exchange under encryption mechanism. That is the parameters are coded before uploading, and the public-private decoding keys are also need to transmit, which cause extra communication cost.

Problems with the Homographic Encryption Model

Problems with the above approach:-

Homographic Encryption somehow deals with the Privacy leak issue or reconstruction issue by not showing the actual parameters to the environment. And the communication Api is also secured by using Authenticated Web tokens i.e. JWT.. etc.

However, It fails to handle these two issues:-

1. **User Dropouts :-** Client failures is one of the major issue in the federated learning which can cause major loss in the accuracy.

2. **Malicious Clients :-** What if the participating clients are malicious itself.

Secret Sharing Technique

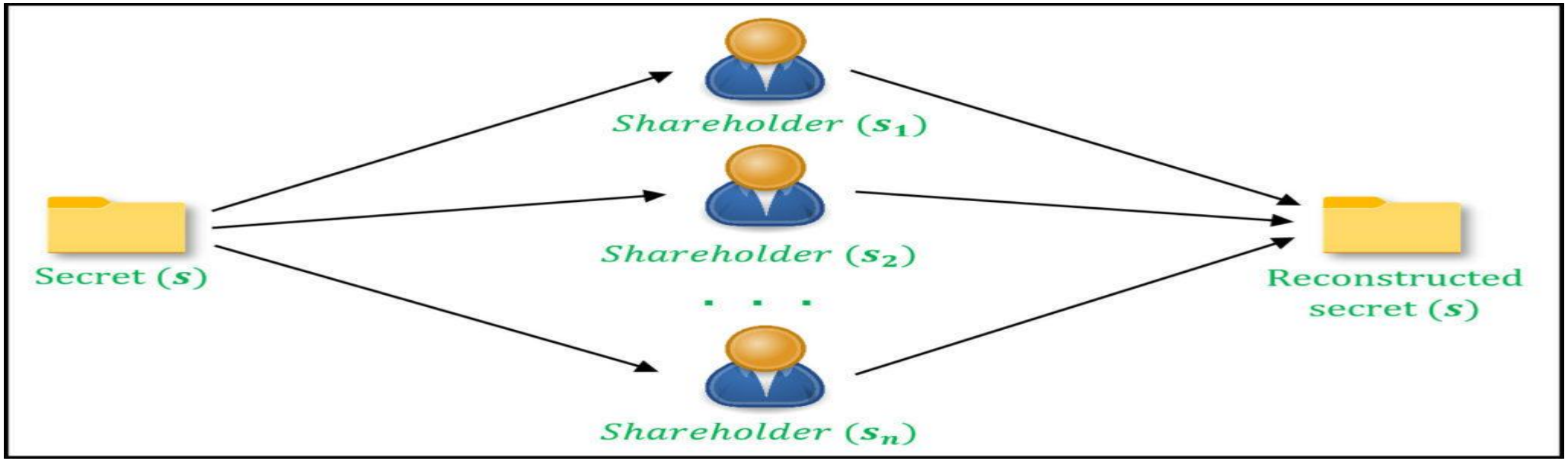
To deal with both of the issues i.e. Client failures and malicious clients, we can use secret sharing technique.

The essence of the primitive is that a dealer wants to split a secret into several shares given to shareholders, in such a way that each individual shareholder learns nothing about the secret, yet if sufficiently many re-combine their shares then the secret can be reconstructed. Intuitively, the question of trust changes from being about the integrity of a single individual to the non-collaboration of several parties: it becomes distributed.

Given a secret S, we would like n parties to share the secret so that the following properties hold:

1) All N parties can get together and recover S
2) Less than n parties cannot recover S

Continued….



As shown in the figure, we have a secret s which is distributed among all shrare-holders as s1, s2, s3 … sn and reconstructed as secret s after the aggregation.

.

Key Steps of Secret Sharing:

- Let's say S is the secret that we wish to encode.
- It is divided into N parts: S1, S2, S3, ..., Sn.
- After dividing it, a number K is chosen by the user in order to decrypt the parts and find the original secret.
- It is chosen in such a way that if we know less than K parts, then we will not be able to find the secret S (i.e.) the secret S can not be reconstructed with (K – 1) parts or fewer.
- If we know K or more parts from S1, S2, S3, ..., Sn, then we can compute/reconstructed our secret code S easily. This is conventionally called (K, N) threshold scheme.

Example of Secret Sharing

- Let the secret code S = 65, N = 4, K = 2..
- Initially, in order to encrypt the secret code, we build a polynomial of degree (K – 1).
- Therefore, let the polynomial be y = a + bx. Here, the constant part ‘a’ is our secret code.
- Let b be any random number, say b = 15.
- Therefore, for this polynomial y = 65 + 15x, we generate N = 4 points from it..
- Let those 4 points be (1, 80), (2, 95), (3, 110), (4, 125).

Example Continued...

- Clearly, we can generate the initial polynomial from any two of these 4 points and in the resulting polynomial, the constant term a is the required secret code.

- In order to reconstruct the given polynomial back, the Lagrange basis Polynomial is used.

- **Lagrange Polynomial**:

$$l_i = \frac{x - x_0}{x_i - x_0} \times \ldots \times \frac{x - x_{i-1}}{x_i - x_{i-1}} \times \frac{x - x_{i+1}}{x_i - x_{i+1}} \times \ldots \times \frac{x - x_{k-1}}{x_i - x_{k-1}}$$

$$f(x) = \sum_{i=0}^{K-1} y_i l_i(x)$$

Example Continued...

- **Re-construction of the secret:-**

$$l_0 = \frac{x - x_1}{x_0 - x_1} = \frac{x - 3}{1 - 3}$$

$$l_1 = \frac{x - x_0}{x_1 - x_0} = \frac{x - 1}{3 - 1}$$

$$f(x) = y_0 l_0 + y_1 l_1$$

$$f(x) = 80\left(\frac{x - 3}{-2}\right) + 110\left(\frac{x - 1}{2}\right)$$

$$f(x) = -40x + 120 + 55x - 55$$

$$f(x) = 15x + 65$$

Iteration Overview in secret sharing

Each Iteration:-

- Initially, the shares of parameters are generated by each client
- These shares are then distributed to all the clients such that each client will have the shares of other clients(a mix-up of shares)
- After client's distribution the parameters are sent to the server for aggregation.
- And then the averaged parameters are broadcasted to all the clients.

How it handles the major issues?

1. **Client Privacy Maintained-** As we are distributing the weights in the network before sending it to the server such that each client has a mix-up of shares of all other clients including some shares of itself. By this distribution, Server don't know which shares belongs to which client.

2. **Dealing with Malicious users-** Iteration involves 2 major steps where the first one is about distributing shares and second is about aggregation.

    - **How distribution done**?:- Let's say we have a system in which 3 clients are participating. And take N = 3 and K = 2 where N is the number of shares generated by each client for a single parameter and K is the minimum no. of shares needed to decrypt the shares.

Continued...

1. **How Distribution Done?** There are 3 clients C1, C2 and C3 each will have 3 shares corresponding to their parameters

   C1:- [11, 12, 13] has 3 shares, C2:- [21, 22, 23] has 3 shares C3:- [31, 32, 33] has 3 shares

   Let's say each client will distribute its one of the share to other clients.

   After Distribution:-

   C1':- [11,21,31], C2':- [12,22,32], C3':- [13,23,33]

   K = 2 which means we need 2 shares of each client to decrypt the parameter but each client has maximum 1 share of the other client.

Continued...

- **Averaging Step**:-

  cavg. = [c1’[i]+c2’[i]+c3’[i]/3] for each i corresponding to each element of

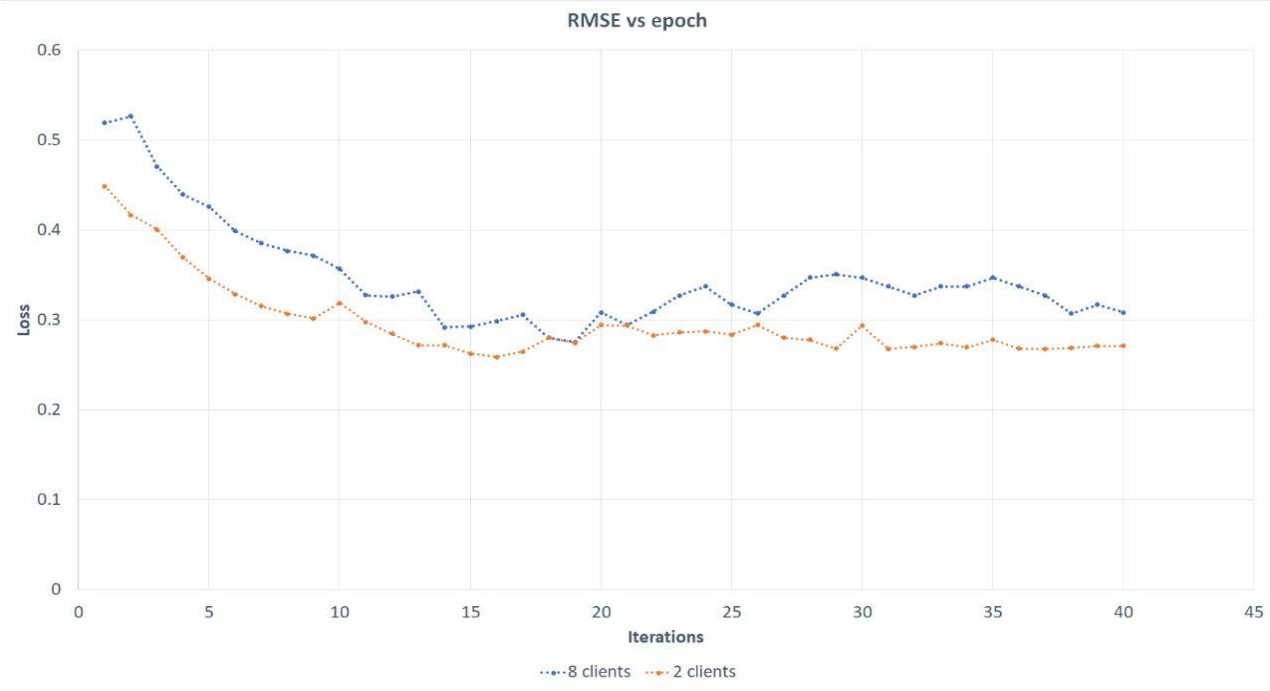
  respective clients

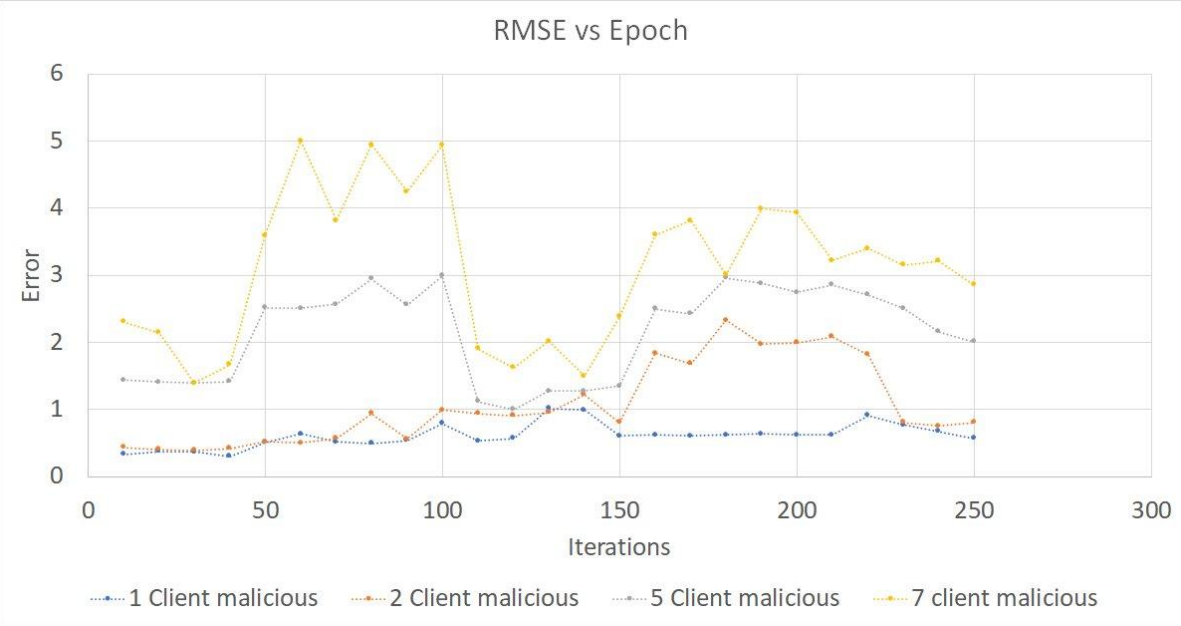
  Then the cavg. is broadcasted and the same iteration follows again.

- **Handle Client failures**:- Let’s say if C1’ lost due to some failure or communication issue i.e. lower bandwidth but we have C1 shares in C2 & C3.

- **Malicious Clients**:- Client C2 can be malicious and try to update the shares of C1 and C3 but again C1, C3 collectively has minimum shares to decrypt.

Observation and *Results(Homographic Encryption)*





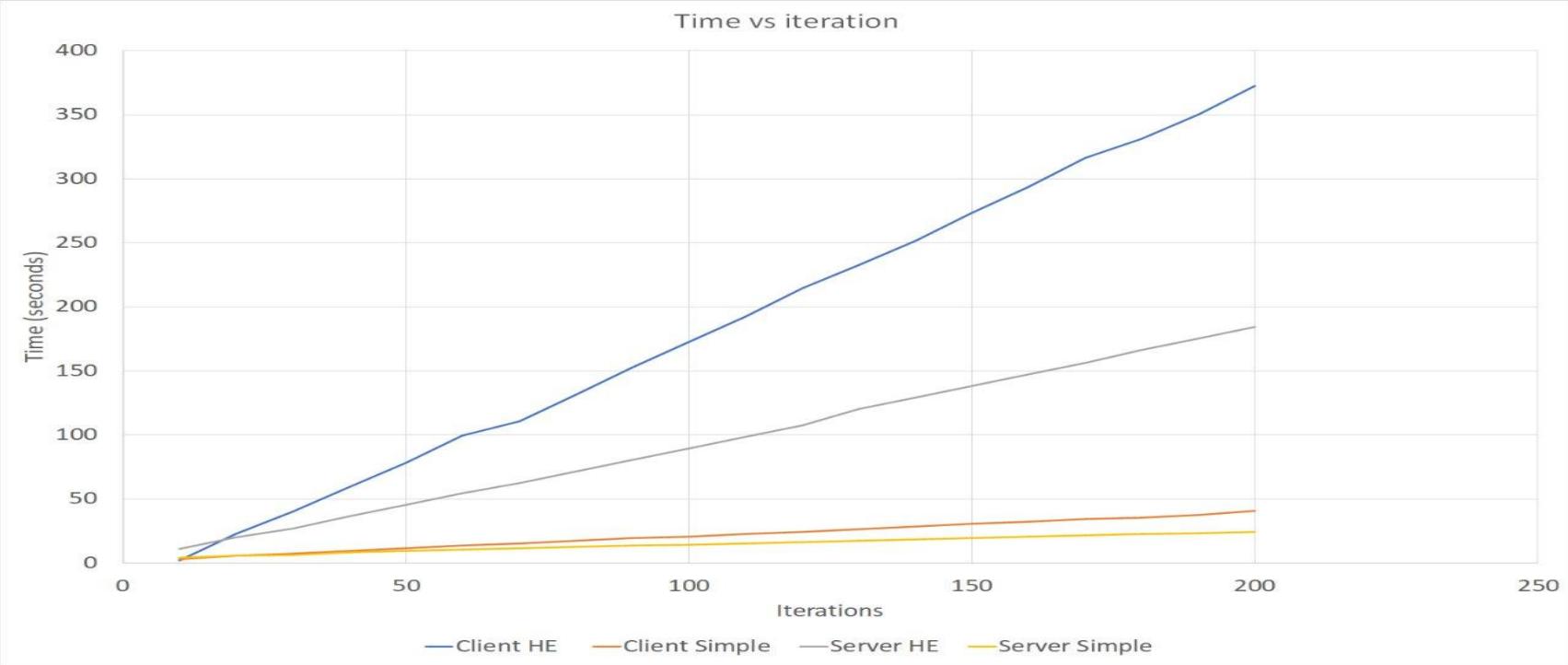
Takeaways

1. RMSE values are converging first and then it's almost constant for 2 clients
2. Some bumps can be seen in case of 8 clients because of the data distribution & participations.

Takeaways

A lot of bumps and variations can be seen in RMSE values in malicious environment because Homographic Encryption works only for the Semi-honest(curious to know) environments.

Observation and *Results(Homographic Encryption)*



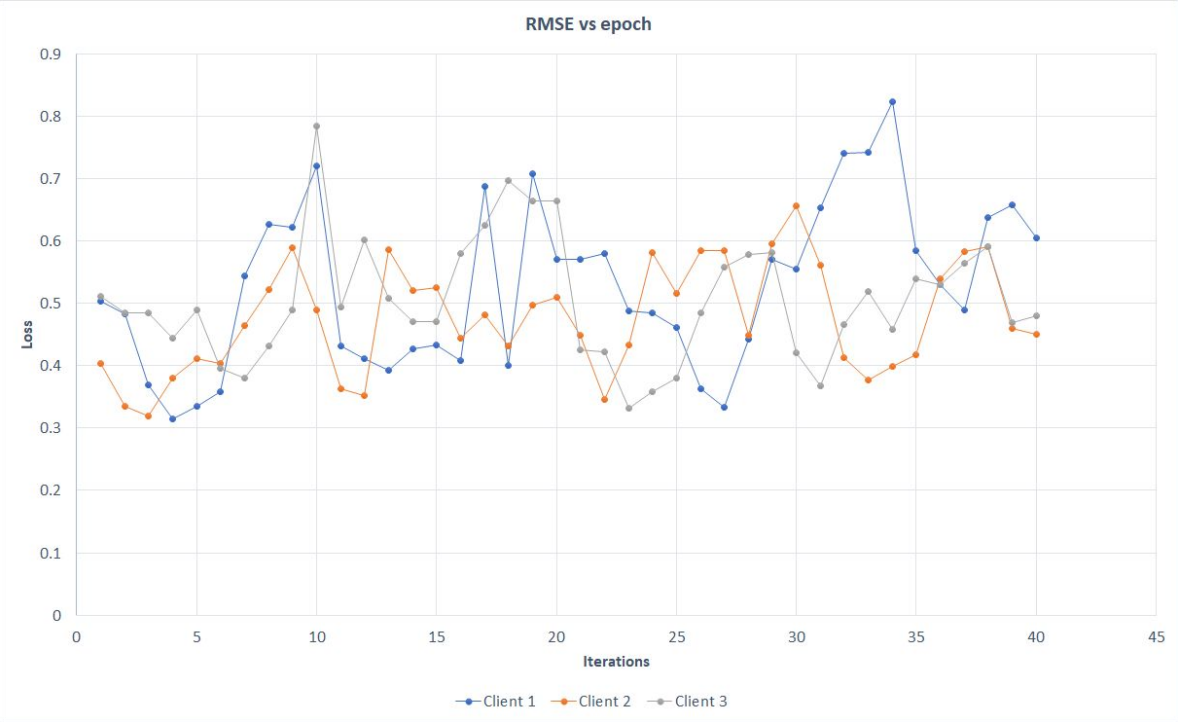
Takeaways

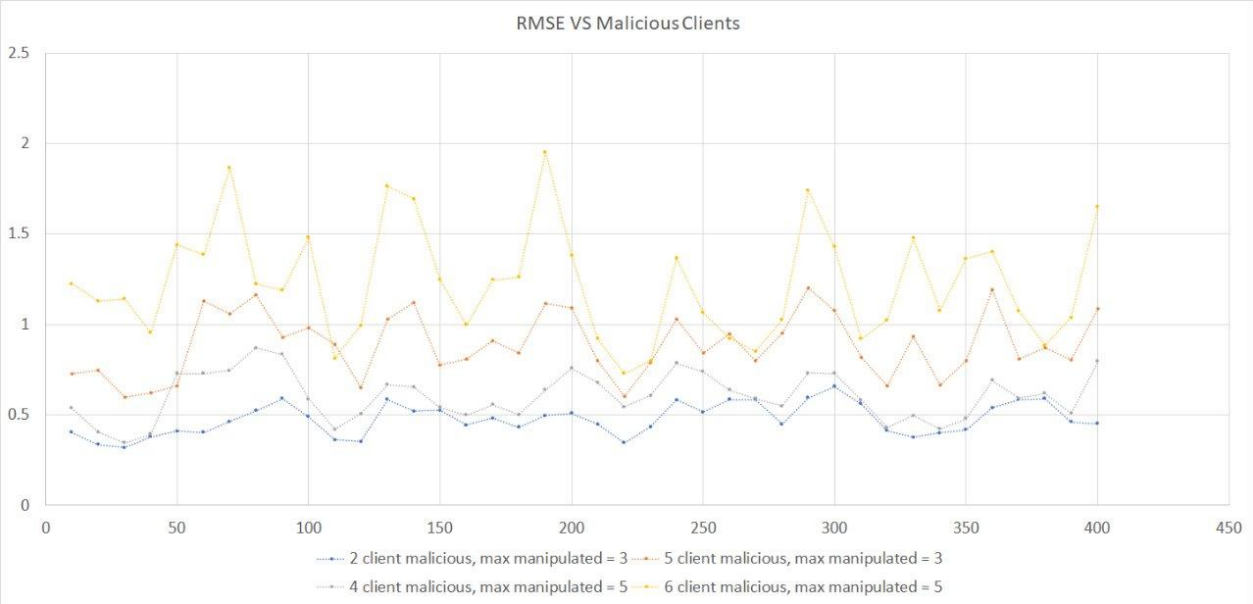
1. The wall-clock running time taken by HE algorithm is very large as compared to simple neural network

Takeaways

2. Encryption and Decryption causes some amount of delay/latency in each iteration w.r.t the neural network model without encryption.

Observation and *Results(Secret Sharing)*





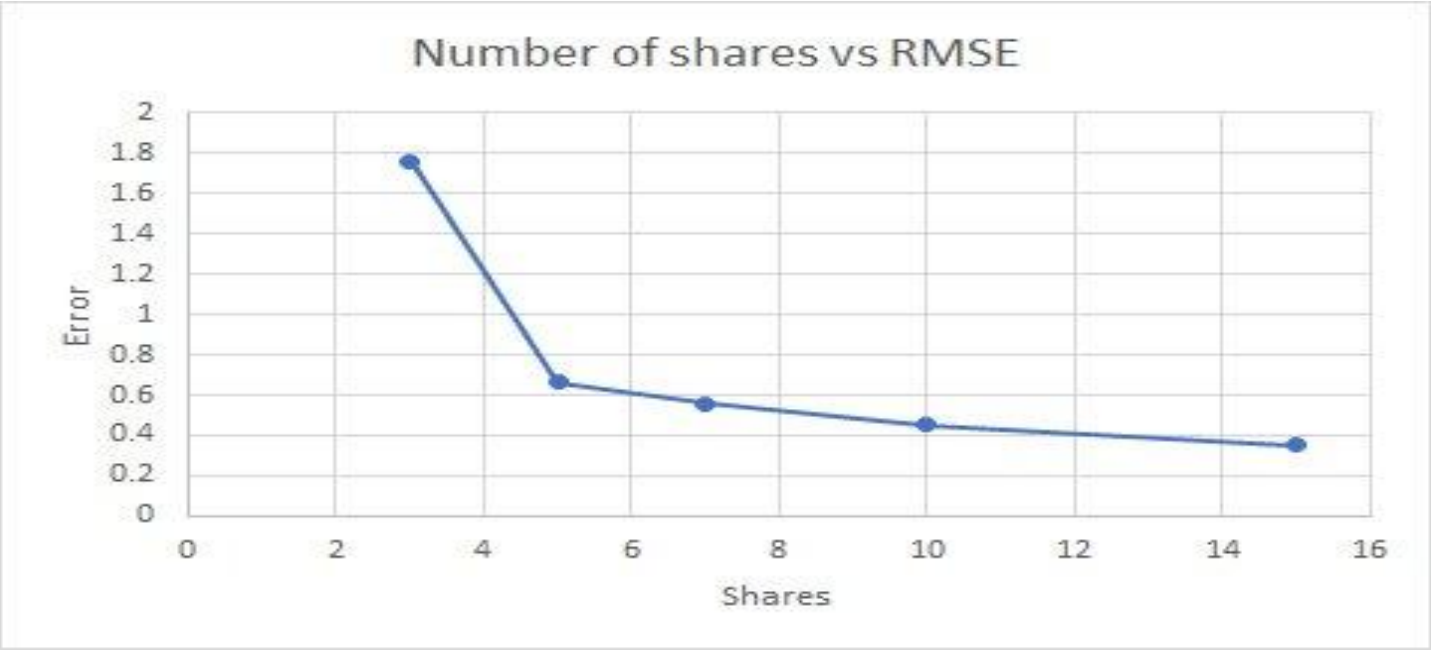
Takeaways

RMSE values are not converging to global optima and many bumps can be seen in the above graph which is because of the taking less precision of weight parameters to deal with high computation.

Takeaways

A sudden bumps and variations can be seen in RMSE values in malicious environment which depends only on our max manipulated parameter which can have a maximum value of N - K.

Observation and *Results(Secret Sharing)*



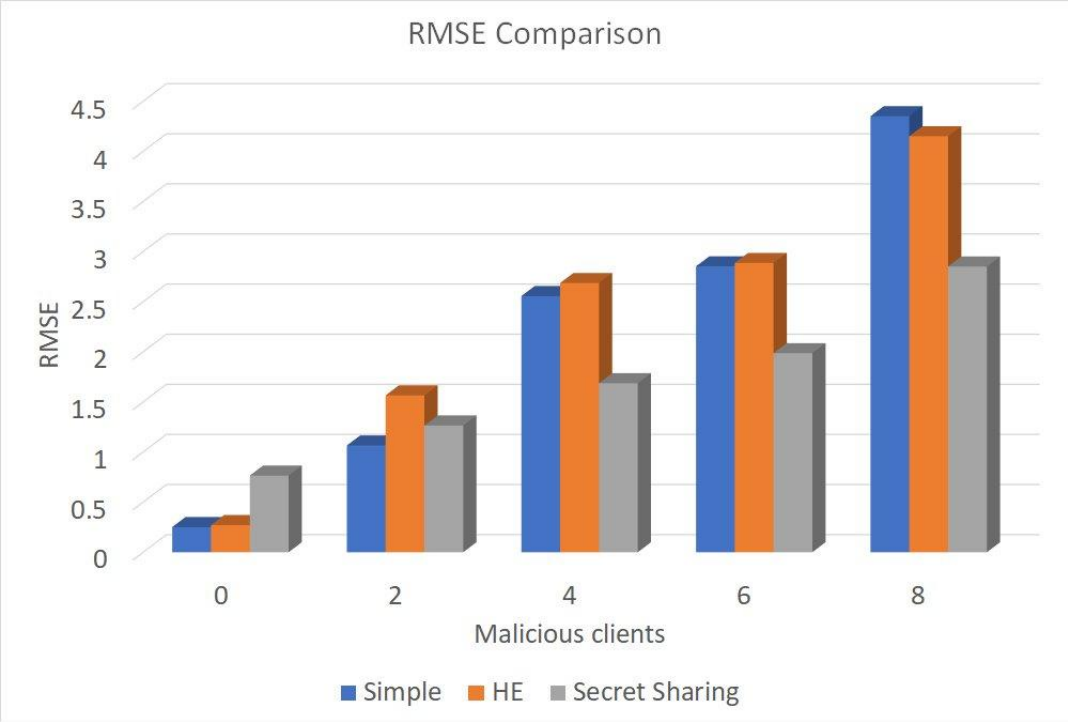
Takeaways

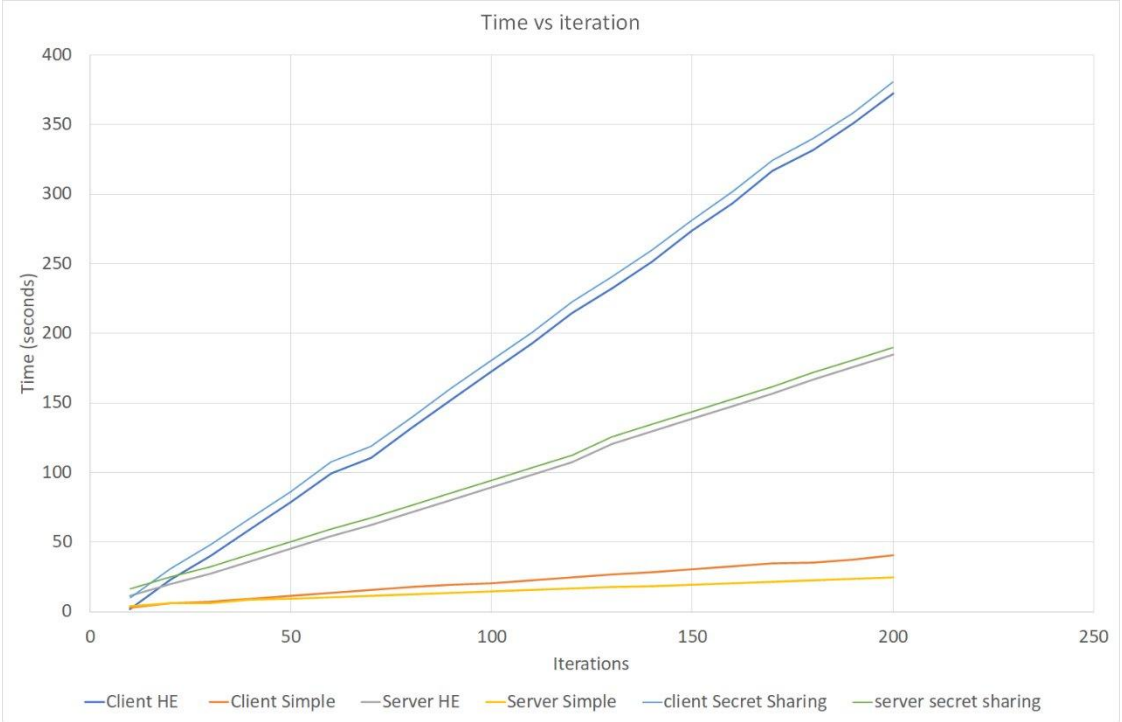
The the RMSE values are decreasing as the number of shares are increasing which is quite obvious because then each client will have a good

Takeaways

mix-up of shares in this case and the malicious clients would not affect the model too much. But the computation time increases so much in this.

Comparison of all three protocols





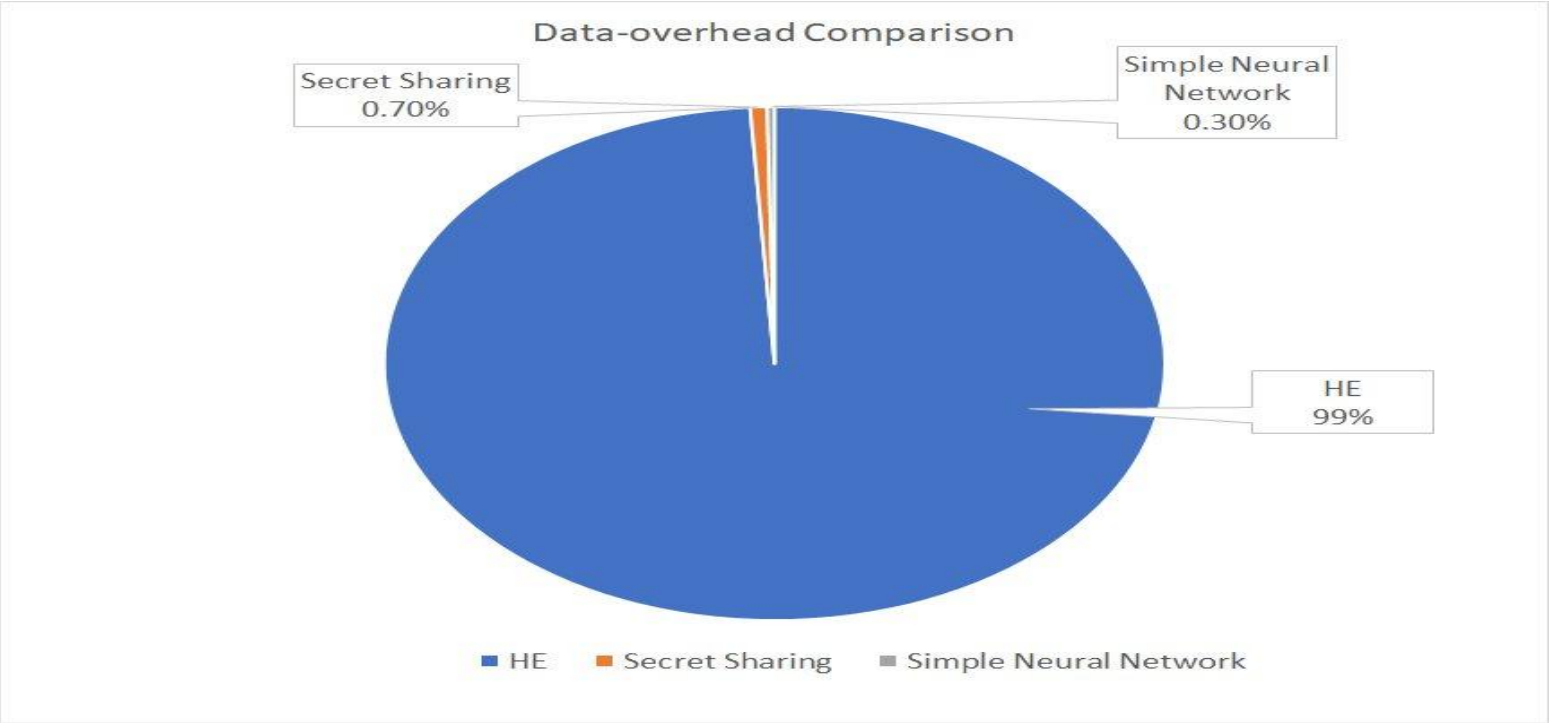
Takeaways

Initially with 0 or 2 malicious clients the Simple and HE model is working better than the secret sharing technique but they are performing worse if the no. of malicious clients are increasing.

Takeaways

The time taken for each iteration by the client and server is almost same for HE & Secret Sharing though it's somewhat large for the Secret Sharing but time taken is very less in case of simple FL.

Continued...



Takeaways

The size of data overhead is too much which is almost(100%) for Homographic Encryption as compared to the other two. As each parameter is encrypted and encoded which has approximate

Takeaways

20000000 bytes(20 MB) length. On the other hand, the data overhead size is almost negligible in case of secret sharing and simple neural network.

Conclusion

1. We have implemented a privacy-preserving federated learning scheme with different cryptographic algorithms i.e. Homographic Encryption and Secret Sharing.

2. Comprehensive experiments were conducted to evaluate the effectiveness and efficiency of both the approaches. Experiment results showed that Secret Sharing made it possible to train the neural network in the malicious environment with some performance loss, and attained computation and communication cost reduction for secure aggregation.

3. However, Homographic encryption and general approach couldn't handle the user dropouts and manipulated users but showing some good and accurate results in terms of performance in the semi-honest system.

What next?

- ★ Good Convergence: By using different techniques i.e. Fast Fourier transform and newton's method to reconstruct shares
- ★ Scalability: Implementing attacks at scale and a scaled distribution network for large no. of clients
- ★ Real world Application: Designing a real world application using the above protocols
- ★ Serverless Architecture: Remove Central Authority

Thank you for listening :)