Decentralized Machine Learning Models with Cryptography Algorithms

Aman, Prof. Siba Narayan Swain, Prof. Mahadeva Prasanna
*BTP Autumn Semester 2020 Report

***Abstract*—Machine learning models trained on sensitive real-world data promise improvements to everything from medical screening to disease outbreak discovery. And the widespread use of mobile devices means even richer—and more sensitive—data is becoming available. However, Traditional machine learning involves a data pipeline that uses a central server(on-prem or cloud) that hosts the trained model in order to make predictions. Distributed Machine Learning (FL) in contrast, 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. However, parameter interaction and the resulting model still might disclose information about the training data used. To address these privacy concerns, two approaches have been used based in this report which is based on Homographic Encryption and Secret Sharing technique among others. The report summarizes the research done in these areas and provides suggestions for future directions in research.**

***Index Terms*—Cryptography, Neural Network, Privacy-preserving, Homographic Encryption, Secret Sharing Scheme.**

I. Introduction

Advancements in machine learning algorithms have resulted in a widespread adoption across industries. Nonetheless, areas dealing with sensitive and private data, like healthcare or finance, have lagged behind due to regulatory constraints to protect users' data. With the emergence of Machine Learning as a Service, entities are providing model inference as a service. We can distinguish three parties in such scenarios, a model owner such as a hospital who has trained a model, a host such as a cloud provider providing computing power, and a client wanting to benefit from the service. A model owner can also be a host in some scenarios. Trust must be established between those parties as the client does not want her data to be leaked, and the model owner wants to protect her model. However, large-scale collection of sensitive data entails risks. At the same time, with the increasing awareness of large companies compromising on data security and user privacy, the emphasis on data privacy and security has become a worldwide major issue. News about leaks on public data are causing great concerns in public media and governments. This work outlines an approach to advancing privacy-preserving machine learning by leveraging secure multiparty computation (MPC) to compute sums of model parameter updates from individual users' devices in a secure manner. The problem of computing a multiparty sum where no party reveals its update in the clear—even to the aggregator—is referred to as Secure Aggregation. The secure aggregation primitive can be used to privately combine the outputs of local machine learning on user devices, in order to update a global model. Training models in this way offers tangible benefits—a user's device can share an update knowing that the service provider will only see that update after it has been averaged with those of other users. The secure aggregation problem has been a rich area of research: different approaches include works based on generic secure multi-party computation protocols, works based on DC-nets, works based on partially- or fully-homographic threshold encryption, and works based on pairwise masking. We discuss these existing works in more detail in Section 9, and compare them to our approach. Mutual research of machine learning and cryptography is not a new area. In addition to cryptography and machine learning has a wide range of applications in relation to information and network security. A none-exhaustive list of examples can be found here:-

1) Network anomaly detection
2) Malware analysis and detection.
3) Homomorphic encrpytion applications
4) Attacks on Physical Unclonable Functions
5) Using machine learning to develop Intrusion Detection System (IDS)
6) Malicious code classification and identification

To address the aforementioned security concerns, and considering the universality nature of neural networks, we explore integration of neural networks with cryptography's techniques to design a secure Decentralized learning system with a semi-honest(curious to know) server. Specifically, we design a secure decentralized based general learning system by transferring encrypted weights over distributed system, which consists of initialization, local weight computation, and global weight aggregation. We are particularly focused on the setting of mobile devices, where communication is extremely expensive, and dropouts are common. Given these constraints, we would like to discuss two popular cryptography techniques i.e. Homographic Encryption, Secret Sharing in a distributed machine learning system and compare the two approaches in terms of loss and performance.

II. Background

In this section, we introduce the background and explain the underlying building blocks i.e. *Federated Learning*, *Homographic Encryption* and *Secret Sharing* of our proposed framework in Figure 1.

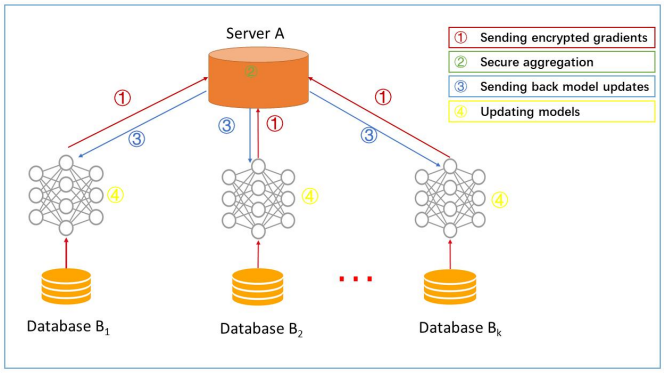

Fig. 1. Architecture of Distributed Machine Learning system and overview of how an iteration works during general federated learning model.

A. Federated Learning

Google has built one of the most secure and robust cloud infrastructures for processing the data to make our services better. Now for models trained from user interaction with mobile devices, we're discussing an additional approach: Federated Learning. Federated Learning (FL) 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.

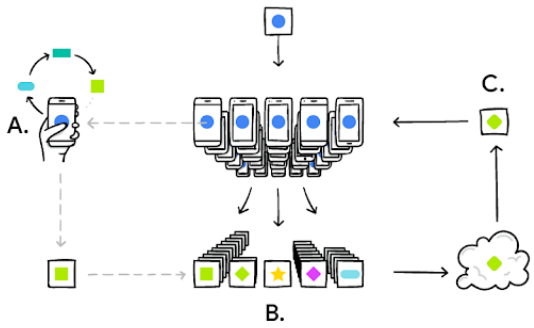

Fig. 2. Your phone personalizes the model locally, based on your usage (A). Many users' updates are aggregated (B) to form a consensus change (C) to the shared model, after which the procedure is repeated

Federated Learning enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on device, decoupling the ability to do machine learning from the need to store the data in the cloud. This goes beyond the use of local models that make predictions on mobile devices (like the Mobile Vision API and On-Device Smart Reply) by bringing model training to the device as well.

Federated Learning allows for smarter models, lower latency, and less power consumption, all while ensuring privacy. And this approach has another immediate benefit: in addition to providing an update to the shared model, the improved model on your phone can also be used immediately, powering experiences personalized by the way you use your phone.

B. 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 her 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 her data, without having shown it once in clear to the third party. It admits some kind of computations on encrypted data. In particular, given an input x encrypted as *Enc(x)*, it should be possible to publicly compute *Enc(x)* for a function f taken from some class of functions. The key point here is "publicly", meaning that carrying out this computation has to be possible without having access to any secret information

C. Secret Sharing

In cryptography, secret sharing refers to any method for distributing a secret among a group of participants, each of which allocates a share of the secret. The secret can only be reconstructed when the shares are combined together; individual shares are of no use on their own.

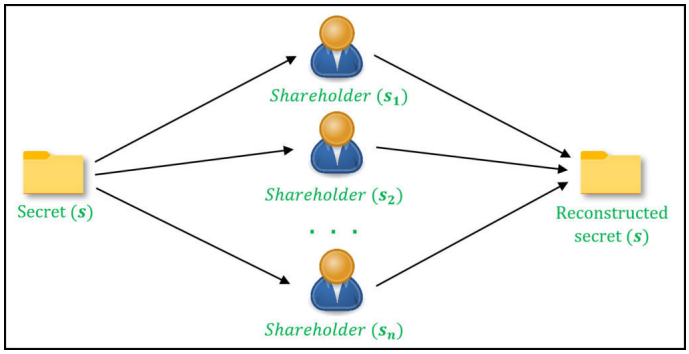

Fig. 3. Secret sharing (also called secret splitting) which refers to methods for distributing a secret among a group of participants, each of whom is allocated a share of the secret is shown in the figure.

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*

As shown in the figure 3, we have a secret *x* which is distributed among all shrareholders *x1, x2, x3 ... xn* and reconstructed as secret *x* after the aggregation.

III. Problem Formulation

In this section, we mathematically formulate the concept of federated learning and then discuss about the problems related to above approach and various threats to the model i.e. *Fault Tolerance*, *Malicious Clients*.

A. Maths behind the Federated Learning

Initially we will serialize the weights present in between the layers of the neural network of client C_p. Let there be $L_1, L_2, ..., L_n$ layers, i.e., if W_1 is a m*n matrix

$$L_1 = W_{1,1}, W_{1,2}, ..., W_{1,n}, W_{2,1}, .., ..., W_{m,1}, W_{m,2}, ...W_{m,n} \tag{1}$$

Similarly, for other layers, we will serialize $L_2, L_3, ..., L_n$ and make a vector

$$L^k = L_1, L_2, ..., L_k \tag{2}$$

representing serialized weights from client C_p.
Now at the server we aggregate the layers from different clients

$$L_{avg} = \frac{1}{P}\sum_{p=1}^{P} L_p \tag{3}$$

where P number of clients send weights to the server.

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. Given that the model's parameters are trained based on the user's data, there is a chance of getting information about the data from such parameters. The updates transmitted from the device to the server should be minimal. There's no need to share more than the minimum amount of info required to update the model at the server. Otherwise, there remains the possibility of private data being exposed and intercepted. The private data is this vulnerable, even without being sent explicitly to the server because it's possible to restore it based on the parameters trained by such data. In the worst case when an attacker is able to restore the data, it should be anonymous as much as possible without revealing some user's private information like the name for example.

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. This creates an opportunity for malicious clients to manipulate the training process with little restriction.

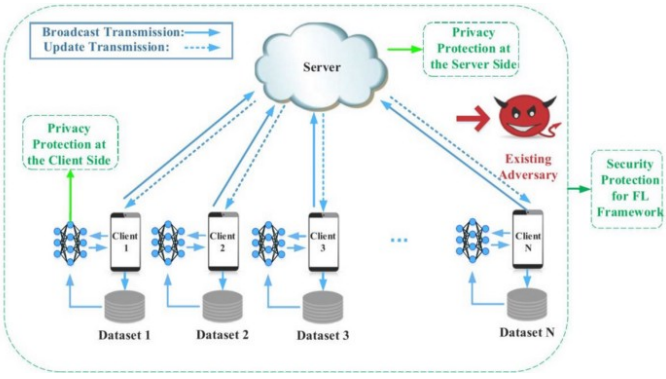

Fig. 4. Malicious client responsible for data poisoning.

Model Aggregation Issue: 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. The main reasons can be concluded as: (i) the noise power of perturbation is increasing along with the number of clients; (ii) the server should know the stochastic information from clients and the design of the aggregation method needs to distinguish the privacy-sensitive clients from privacy insensitive ones.

IV. Experimental Results and Possible Solutions

In this section, we discuss about the possible solutions and some experimental results to solve the above problems.

A. Homographic Encryption with Authentication for privacy leak

As discussed in Section 2, After the client trains the model by its private data, the model is sent to the server. At this time, an attacker might hack some of the Apis of the model to make it behave for their benefit. For example, the attacker might control the labels assigned to images with certain features. As for the security of the whole FL framework, it mainly considers the model stealing attacks. Specially, any participant in FL may introduce hidden functionality into the joint global model, e.g., to ensure that an image classifier assigns an attacker-chosen label to images with certain features, or that a word predictor completes certain sentences with an attacker. Consequently, there are also some protecting measures on the security design for FL. To handle these issues, we used Homographic Encryption with Authentication such that even after sharing the parameters to server or to any other client, it can't be decrypted and the privacy of each of the client is maintained.

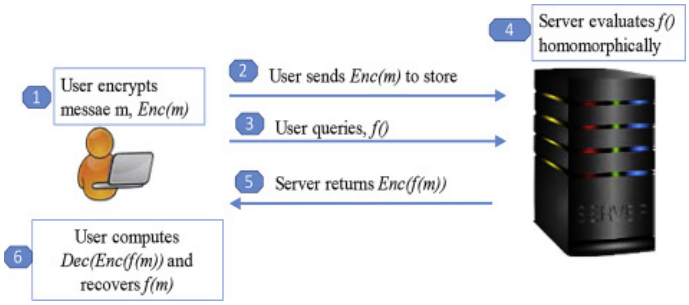

Fig. 5. Homographically Encrypted Training.

Homomorphic encryption [12] 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.

Key Steps of Homographic Encryption:

Let P be the plaintext space, i.e., P = {0,1} which consists of

input message tuple (m1, m2,..mn). Let us represent the Boolean circuit by C and ordinary function notation as C (m1, m2,... mn) to represent the evaluation of the circuit on the message tuple . The general HE is described below:

- Gen(1λ, α) is the key generation algorithm that generates output keys triplets, i.e., secret key-pair (sk and pk) along with evaluation key (evk), where λ is security parameter and α is auxiliary input, (sk, pk, evk) $\leftarrow$ KeyGen()
- Enc(pk,m) encrypts a message (m) with the public key (pk) and outputs a ciphertext (c ϵ C), c $\leftarrow$ Encpk(m)
- Dec(sk, c) decrypts a ciphertexts with the secret key (sk) and recovers message (m) as the output, m $\leftarrow$ Decsk(c)
- Eval(evk, C, c1, c1, ..., cn) produces evaluation output by taking evk key as input, a circuit CϵC and tuple of input ciphertexts, i.e., c1...cn and previous evaluation results, c $\leftarrow$ Evalevk(evk, C, c1, c2, ...cn).

Observation and Results:

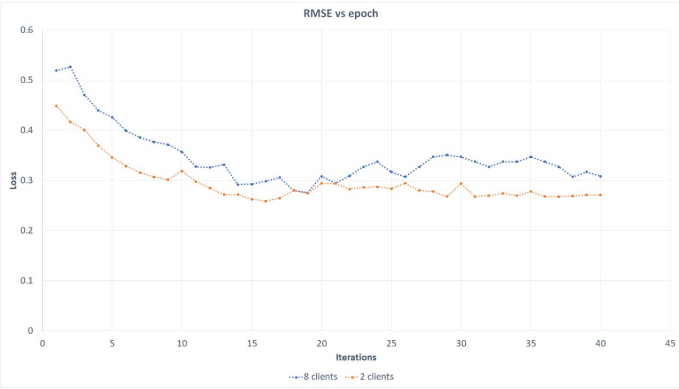

Fig. 6. RMSE values variation with number of Iterations in Homographic Encryption for two and eight clients.

As shown in the **figure 6**, we can observe that RMSE values for two clients are decreasing and the model is converging first and then it's almost constant which is expected according to the theoretical aspects. Also, it shows some variation and bumps in case of 8 clients which is happening because of the more data distribution and participation of more clients in aggregation.

In the **figure 7**, the wall-clock running time taken by HE algorithm is very large as compared to simple neural network which causes some amount of delay or latency in each iteration w.r.t the neural network without encryption.

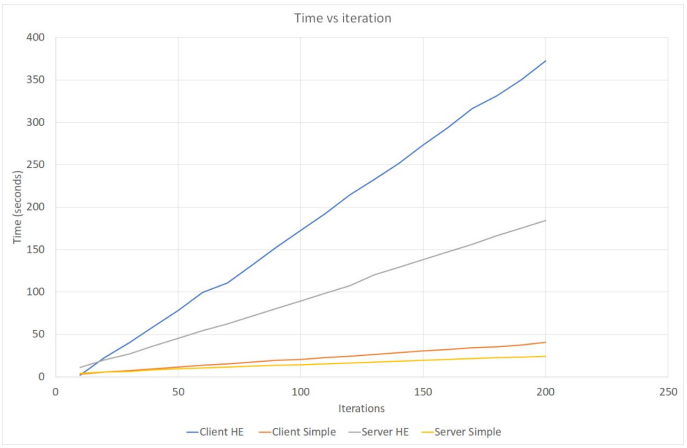

Fig. 7. Wall-clock running time for the client and server with and without Homographic Encryption in each iteration.

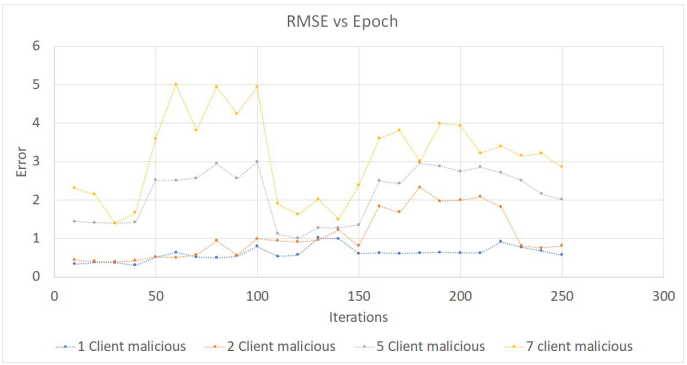

Fig. 8. RMSE values variation with number of Iterations for malicious clients

we can observe a lot of bumps and variations in RMSE values in **figure 8** because Homographic Encryption works only for the Semi-honest(curious to know) environment where all the clients and server wants to know each other's parameters but they can't manipulate it. And it's not possible in some scenario. So, we discuss about Secret Sharing technique which can handle all of the above issues to some extent.

B. Secret Sharing for reducing Data Poisoning and improving Model Aggregation

As discussed in section 3, Federated learning (FL) is an emerging paradigm for distributed training of large-scale deep neural networks in which participants' data remains on their own devices with only model updates being shared with a central server. However, the distributed nature of FL gives rise to new threats caused by potentially malicious participants. To overcome this issue we see about the secret sharing technique. Secret sharing is an old well-known cryptographic primitive, with existing real-world applications in e.g. Bitcoin signatures and password management. But perhaps more interestingly, secret sharing also has strong links to secure computation and

may for instance be used for private machine learning.

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.

Key Steps of Secret Sharing: The main aim of this algorithm is to divide secret that needs to be encrypted into various unique parts.

- 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.

Approach: The main idea behind the Secret Sharing Algorithm lies behind the concept that for the given K points we can find a polynomial equation with the degree (K – 1).

Example: For the given two points, (x1, y1) and (x2, y2) we can find a linear polynomial ax + by = c. Similarly, for the given three points, we can find a quadratic polynomial ax2 + bx + cy = d.

So, the idea is to build a polynomial with the degree (K – 1) such that the constant term is the secret code and the remaining numbers are random and this constant term can be found by using any K points out of N points generated from this polynomial by using Lagrange's Basis Polynomial.

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..

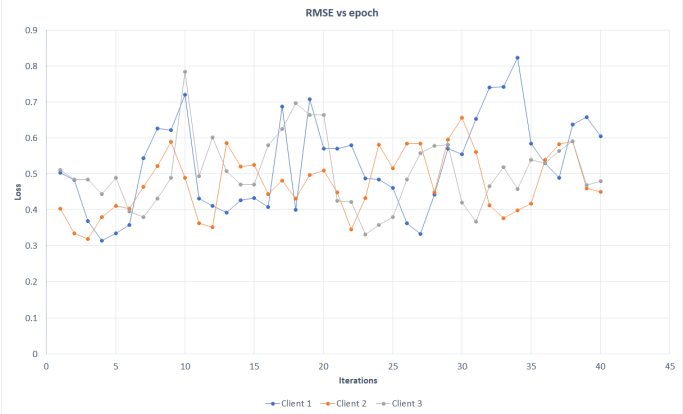

Fig. 9. RMSE values variation with number of Iterations for three clients

- Let those 4 points be (1, 80), (2, 95), (3, 110), (4, 125). 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

The concept behind the Lagrange polynomial[12] is to form the Lagrange's identities first and the summation of these identities give us the required function which we need to find from the given points.

Iteration overview in secret sharing protocol:-

- 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.
- After client's distribution the parameters are sent to the server for aggregation.
- And then the averaged parameters are broadcasted to all the clients.

Observations in secret sharing protocol:-

As shown in the **figure 9**, we can see that there is a lot of variation in the RMSE curves as compared to the figure 7, it's not converging and not reaching to global optima. It's happening because of the precision value of the shared parameters which is 3 in case of secret sharing and weights are changing so fast as compared to simple neural network which causes high and low RMSE sometimes. And if we are taking the high precision it's time complexity is increasing as it has many steps i.e. generate shares, distribute shares, model aggregating and reconstructing the shares using Lagrange interpolation and it needs high computational power CPU which shows us a trade-off between the accuracy vs resources required.

Fig. 10. RMSE values variation with number of Iterations for malicious clients

As shown in the **figure 10**, we can observe that there is a sudden bump or increase in the RMSE values curves as compared to the figure 1 when no. of malicious clients increases and no. of shares, max Failures are kept constant and max manipulated is N-K shares which means we need to have at least K shares to reconstruct the secret where N is the no. of shares generated and K is the no. of shares required to decrypt the secret. And we can successfully do an aggregation if we have atleast k shares at the server.

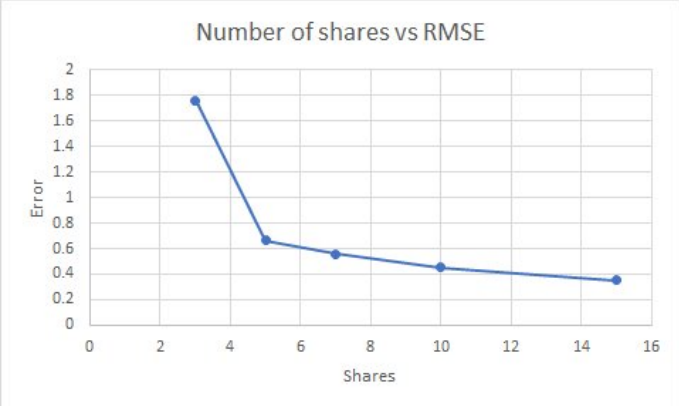

Fig. 11. RMSE variation with number of shares used

As shown in the **figure 11**, the RMSE values are decreasing as the number of shares are increasing which is quite obvious because then each client will have a good mix-up of shares in this case and the malicious clients would not affect the model too much. But the computation time increases for the large number of shares and we need more powerful resources to prevent our model from malicious clients.

C. Comparison between the three protocols:

RMSE Comparison:

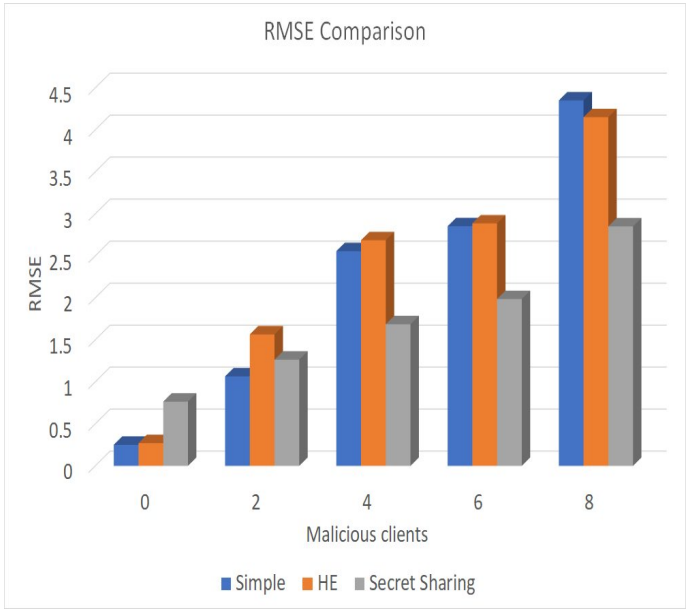

Fig. 12. RMSE comparison for all 3 protocols

As shown in the **figure 12**, Initially with 0 or 2 malicious clients the Simple neural network and Homographic Encryption model is better than the secret sharing technique but it is performing worse if the no. of malicious clients are increasing which shows the secret sharing is better in a scenario where there are large number of malicious clients. It depends upon the situation of the environment from which we can decide the model one should go with.

Time Comparison:

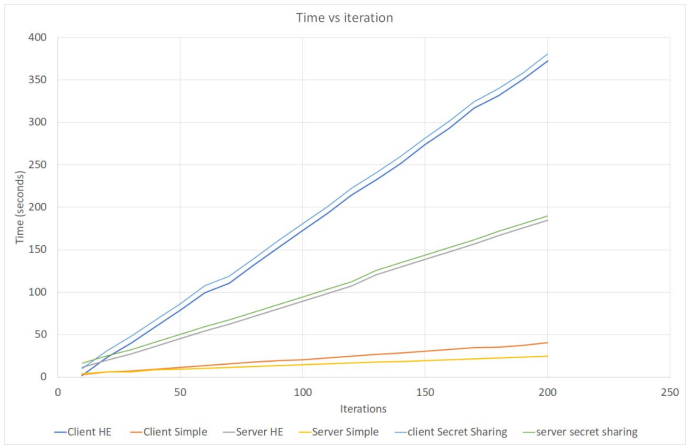

Fig. 13. Wall-clock running time comparison for all 3 protocols

Data-overhead Size Comparison:

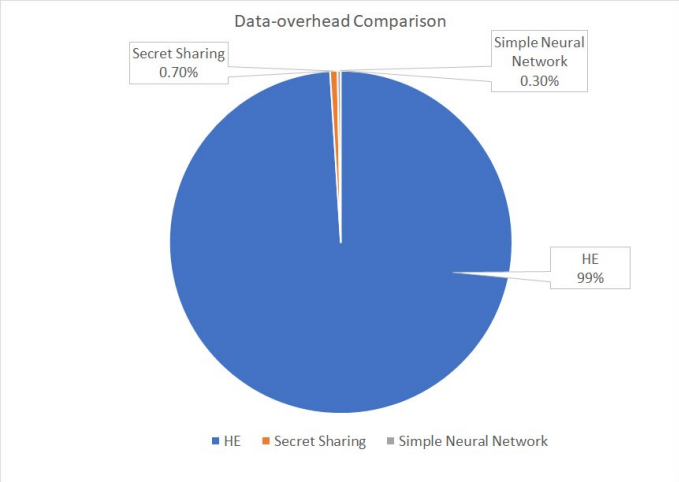

Fig. 14. Data-overhead comparison for all 3 protocols

As shown in the **figure 13**, the time taken for each iteration by the client and server is almost same for Homographic Encryption and Secret Sharing though it's value is somewhat large for Secret Sharing technique. And the time consumed in case of simple neural network is small which shows the trade-off between the privacy and computation time.

As shown in the **figure 14**, 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 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.

V. Conclusion

In this report, we implemented a privacy-preserving federated learning scheme using different cryptographic algorithms i.e. Homographic Encryption and Secret Sharing. We also discussed the privacy related issues and threats to model in a general federated learning approach and tried to use the homographic encryption and secret sharing techniques, which can force the central server to conduct the aggregation operation, and is robust against user dropouts and malicious users. Then, using the designed secure aggregation scheme, we designed a suite of secure protocols to implemented the distributed Neural network linear regression model. 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 and manipulated environment with some performance loss, and attained computation and communication cost reduction for secure aggregation. 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 and the data-overhead of communication was too large in case of Homographic Encryption.

VI. Future Work

We have looked at both the cryptography algorithms in which **secret sharing** was showing some promising results in terms of privacy. But we actually need to improve the performance of the system and try to reduce the loss and time complexity by following ways:-

1) Various approaches for reconstructing the secret i.e. Fast Fourier Transform and Newton's method which can improve the time complexity and hence precision.
2) Scaling and designing of the distributed systems for large number of clients
3) Design a System with mixed approach of Homographic Encryption and Secret Sharing
4) Reach to **Serverless** architecture by removing central authority
5) Checking beats of all the instances using multi-threading

References

[1] Ronald L. Rivest, "Cryptography and machine learning" *Supported by NSF grant CCR-8914428, ARO grant N00014-89-J-1988, and the Siemens Corporation*

[2] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar and Heiko Ludwig"HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning", *ACM ISBN 978-1-4503-6833-9/19/11. . .*

[3] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, Blaise Aguera y Arcas *"Communication-Efficient Learning of Deep Networks from Decentralized Data"*, Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

[4] Brendan McMahan and Daniel Ramage "Federated Learning: Collaborative Machine Learning without Centralized Training Data", https://ai.googleblog.com/2017/04/federated-learning-collaborative.html Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

[5] Qinbin Li, Zeyi Wen, Zhaomin Wu, Sixu Hu, Naibo Wang, Bingsheng He, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection", *arXiv:1907.09693v4 [cs.LG] 1 Apr 2020*

[6] Michele Minelli, "Fully homomorphic encryption for machine learning", "Cryptography and Security [cs.CR]" *Université Paris sciences et lettres, 2018. English. ffNNT : 2018PSLEE056ff. fftel-01918263v2f*

[7] Lidong Zhou, "Secret Sharing", CS Cornell, USA http://www.cs.cornell.edu/courses/cs513/2000SP/SecretSharing.html

[8] Maeva Benoit and Morten Dahl, "How Practical is Somewhat Homomorphic Encryption Today?", https://medium.com/snips-ai/how-practical-is-somewhat-homomorphic-encryption-today-6818d1c6f7f6

[9] Morten Dahl, "High-Volume Secret Sharing" https://medium.com/snips-ai/optimizing-threshold-secret-sharing-c877901231e5

[10] Yang Liu Zhuo Ma, Ximeng Liu, Siqi Ma, Surya Nepal, Robert.H Deng, Kui Ren, "Boosting Privately: Federated Extreme Gradient Boosting for Mobile Crowdsensing" *arXiv:1907.10218v2 [cs.CR] 10 Apr 2020*

[11] Secret double octopus, "The Secret Security Wiki" https://doubleoctopus.com/security-wiki/encryption-and-cryptography/secret-sharing/

[12] Wikipedia the free encyclopedia, "Lagrange polynomial" https://en.wikipedia.org/wiki/Lagrange$_{p}olynomial$

[13] Chuan Ma , Jun Li, Ming Ding, Howard H. Yang, Feng Shu, Tony Q. S. Quek, H. Vincent Poor "On Safeguarding Privacy and Security in the Framework of Federated Learning" *arXiv:1909.06512 [cs.NI]*

[14] Ahmed Gad "Breaking Privacy in Federated Learning" https://heartbeat.fritz.ai/breaking-privacy-in-federated-learning-77fa08ccac9a

[15] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, Michael Moeller, "Inverting Gradients – How easy is it to break privacy in federated learning?" *arXiv:2003.14053 [cs.CV]*

[16] Daniel Huynh, "Homomorphic Encryption intro: Part 1: Overview and use cases" https://towardsdatascience.com/homomorphic-encryption-intro-part-1-overview-and-use-cases-a601adcff06c

[17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone†‡,H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal and Karn Seth, "Practical Secure Aggregation for Privacy-Preserving Machine Learning" *Cornell Tech, 2 West Loop Rd., New York, NY 10044*