Mohit Sharma, et. al. International Journal of Engineering Research and Applications
www.ijera.com
ISSN: 2248-9622, Vol. 12, Issue 4, (Series-IV) April 2022, pp. 58-65

RESEARCH ARTICLE OPEN ACCESS

Decentralized Machine Learning Models with Cryptographic Techniques

Mohit Sharma, Sanchita Shirur**, Anurag Singh***, Prof Preeti Satao*****
** (Department of Computer Engineering, MCTs Rajiv Gandhi Institute of Technology, India*
***(Department of Computer Engineering, MCTs Rajiv Gandhi Institute of Technology, India*
****(Department of Computer Engineering, MCTs Rajiv Gandhi Institute of Technology, India*
****(Department of Computer Engineering, MCTs Rajiv Gandhi Institute of Technology, India*

ABSTRACT
Everything from medical screening to disease outbreak detection could benefit from machine learning models trained on sensitive real-world data. And, thanks to the widespread use of mobile devices, even more detailed—and sensitive—information is becoming available. Traditional machine learning, on the other hand, involves a data pipeline that uses a central server (on-premises or in the cloud) to host the trained model and make predictions. Distributed Machine Learning (FL), on the other hand, is a method of downloading the current model and computing an updated model using local data at the device itself (a.k.a. edge computing).These locally trained models are then sent back to the central server, where they are aggregated (i.e. weights are averaged), and a single consolidated and improved global model is then sent back to the devices. The interaction of parameters and the resulting model, however, may still reveal information about the training data used. Two approaches have been used in this report to address these privacy concerns, which are based on Homographic Encryption and Secret Sharing techniques, among others. The report summarises previous research in these areas and makes recommendations for future research.

Keywords - Cryptography, Neural Network, Privacy- preserving, Homographic Encryption, Secret Sharing Scheme

---

Date of Submission: 13-04-2022 Date of Acceptance: 29-04-2022

I. INTRODUCTION

Machine learning algorithms have advanced to the point that they are now widely used across sectors. Nonetheless, industries that deal with sensitive and private data, such as healthcare and banking, have lagged behind due to legislative requirements to protect consumers' information. Entities are now offering model inference as a service, thanks to the rise of Machine Learning as a Service. In such cases, we may differentiate three parties: a model owner, such as a hospital that has trained a model, a host, such as a cloud provider that provides computing resources, and a client who wants to use the service. In rare cases, a model owner might also be a host.Because the client does not want her data exposed and the model owner wants to protect her model, trust must be developed between the two sides. Large-scale acquisition of sensitive data, on the other hand, poses hazards. At the same time, as huge corporations become more conscious of the dangers of compromising data security and user privacy, the importance of data privacy and security has become a global concern. The news of data leaks is raising tremendous anxiety in the public media and among governments. This paper describes a method for increasing privacy-preserving machine learning by using secure multiparty computing (MPC) to securely compute sums of model parameter updates from individual users' devices.Secure Aggregation is the problem of computing a multiparty sum where no party reveals its update in the open, even to the aggregator. To update a global model, the secure aggregation primitive can be used to privately combine the outputs of local machine learning on user devices. This type of training model has real-world applications: a user's device can share an update knowing that the service provider will only see it after it has been averaged with the updates of other users. The secure aggregation problem has attracted a lot of attention: there have been works based on generic secure multi-party computation protocols, DC-nets, partially- or fully-homographic threshold

encryption, and pairwise masking, to name a few. In Section 9, we go over these previous works in greater depth and compare them to our approach. Mutual machine learning and cryptography research is not a new field. Machine learning, in addition to cryptography, has a wide range of applications in information and network security. The following is a non-exhaustive list of examples:-

1) Detection of network anomalies

2) Malware detection and analysis

3) Applications of homomorphic encrpytion

4) Physical Unclonable Functions Attacks

5) Developing an intrusion detection system using machine learning (IDS)

6) Classification and identification of malicious codes

To address the aforementioned security concerns, we investigate the integration of neural networks with cryptography techniques to design a secure Decentralized learning system with a semi-honest(curious to know) server, taking into account the universality nature of neural networks. We design a secure decentralised based general learning system that consists of initialization, local weight computation, and global weight aggregation, using transfer-ring encrypted weights distributed over a distributed system. We're especially interested in the context of mobile devices, where communication is extremely costly and dropouts are common. Given these constraints, we'd like to compare the loss and performance of two popular cryptography techniques, namely Homographic Encryption and Secret Sharing, in a distributed machine learning system.

II. BACKGROUND

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

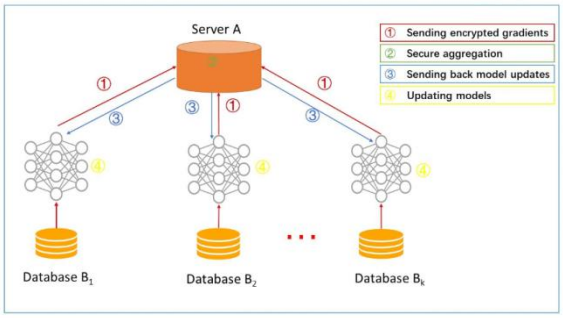

Fig. 1. Architecture of Distributed Machine Learning system and overview of how an iteration works during general federated learning model.

A. Federated Learning

For processing data to improve our services, Google has built one of the most secure and robust cloud infrastructures available. We're now discussing a new approach for models trained from user interaction with mobile devices: Federated Learning. Federated Learning (FL) is a technique that downloads the current model and uses local data to compute an updated model on the device itself (a.k.a. edge computing). These locally trained models are then sent back to the central server, where they are aggregated (i.e. weights are averaged), and a single consolidated and improved global model is then sent back to the devices.

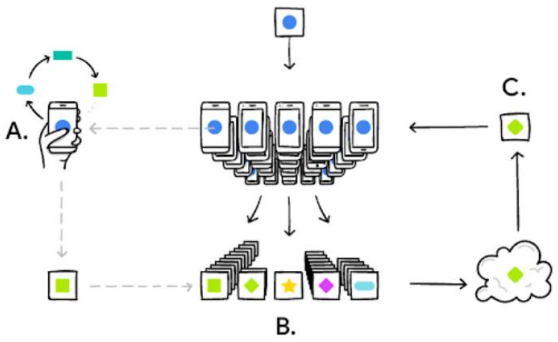

Fig. 2. Your phone personalizes the model locally, based on your usage (A). Many users' updates are aggregated (B) to form a consensus change (C) to the shared model, after which the procedure is repeated

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

Federated Learning enables smarter models, faster response times, and lower power consumption while maintaining privacy.

This approach also has another immediate benefit: in addition to providing an update to the shared model, the improved model on your phone can be used right away, allowing you to create experiences tailored to your preferences.

B. Homomorphic Encryption

Homographic Encryption is a type of encryption that is used to encrypt data. Homographic Encryption is a cryptographic scheme that uses public keys. The user generates a secret and public key pair, then encrypts her data with the public key before sending it to a third party who will perform computations on it. Because of the homomorphic properties of encryption and decryption, the user can obtain the encrypted result and decode it with her own key to view the output of the computation on her data without having to show it to a third party in clear. It allows some computations to be performed on encrypted data. For example, given an encrypted input x, it should be possible to publicly compute Enc(x) for a function f from a class of functions.he key word here is "publicly," which means that this computation must be possible without requiring access to any confidential information.

C. Secret Sharing

Secret sharing is a term used in cryptography to describe any method for distributing a secret among a group of participants, each of whom receives a share of the secret. Only by combining the shares can the secret be reconstructed; individual shares are useless on their own.

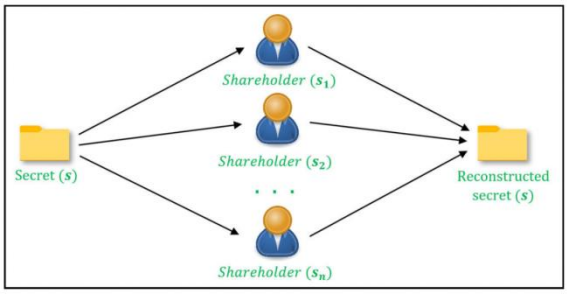

Fig. 3. Secret sharing (also called secret splitting) which refers to methods for distributing a secret among a group of participants, each of whom is allocated a share of the secret is shown in the figure.

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 shareholders x1, x2, x3 ... xn and reconstructed as secret x after the aggregation.
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

$$L_1 = W_{1,1}, W_{1,2}, ..., W_{1,n}, W_{2,1}, .., ..., W_{m,1}, W_{m,2}, ...W_{m,n} \quad (1)$$

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

$$L^k = L_1, L_2, ..., L_k \quad (2)$$

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

$$L_{avg} = \frac{1}{P}\sum_{p=1}^{P} L_p \quad (3)$$

where P number of clients send weights to the server.

Privacy Leak: A problem caused by insecure communication in which the privacy of the clients' shared parameters is compromised, and the server learns about the shared weights or parameters. Given that the model's parameters are trained using the user's data, there is a chance that such parameters can provide information about the data. The number of updates sent from the device to the server should be kept to a minimum. There's no need to share any more information than is necessary to update the model on the server. Otherwise, there's always the risk of personal information being exposed and intercepted.Even if private data is not sent explicitly to the server, it is vulnerable because it is possible to restore it using the parameters trained by such data. In the worst-case scenario, if an attacker is able to recover the data, it should be as anonymous as possible, avoiding revealing personal information such as a user's name.

Data Poisoning: In FL, clients can now view intermediate model states and contribute arbitrary changes as part of the decentralized training process, whereas before they could only act as passive data providers. Malicious clients will be able to alter the training process with little limitation as a result of this.

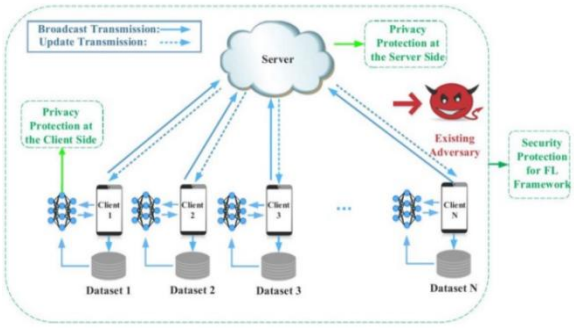

Fig. 4. Malicious client responsible for data poisoning

Model Aggregate Problem: After collecting individual parameters, the aggregation is mostly done on the server, which then updates the global model. This procedure is crucial since it must take into account the clients' benefits and define the end of the learning process. When a client-side protection measure is used, such as perturbation before collecting model parameters, the aggregation procedure cannot be as simple as averaging. The primary reasons are as follows: The number of clients increases the noise power of perturbation; (ii) the server must know the stochastic information from clients, and the aggregation method must distinguish between privacy-sensitive and privacy-insensitive clients.

III. EXPERIMENTAL RESULTS AND POSSIBLE SOLUTIONS

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

A. Homographic Encryption with Authentication for privacy leak

The model is delivered to the server after the client trains it using its private data, as detailed in Section 2. At this point, an attacker may manipulate parts of the model's Apis to make it act in their favor. For example, the attacker could have influence over the labels that are allocated to photographs with specific characteristics. The FL framework's overall security is primarily concerned with model theft attacks. In particular, any FL member may inject hidden functionality into the joint global model, such as ensuring that an image classifier assigns an attacker-selected label to photos with certain attributes or that a word predictor completes certain sentences with an attacker. As a result, various safeguards are included in the security design for FL. To address these concerns, we employed Homographic Encryption with Authentication, which ensures that even after sharing parameters with the server or another client, they cannot be decoded, ensuring that each client's privacy is protected.

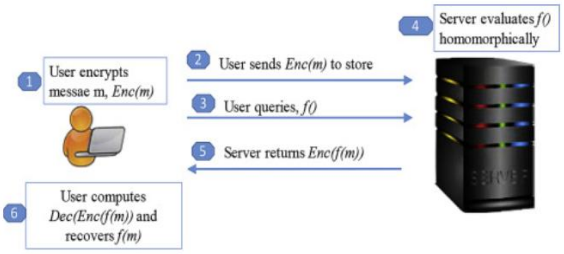

Fig. 5. Homomorphically Encrypted Training.

Homomorphic encryption is used to protect user data by exchanging parameters throughout the encryption process. That is, the parameters must be encoded before being uploaded, and the public-private decoding keys must also be sent, incurring additional communication costs.

Key Steps of Homographic Encryption:

Let P denote the plaintext space, which is defined as P = 0,1 and comprises input message tuples (m1, m2,..mn). To describe the circuit's evaluation on the message tuple, let's use the letter C and the conventional function notation C (m1, m2,... mn).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) KeyGen()

• Enc(pk,m) encrypts a message (m) with the public key (pk) and outputs a ciphertext (c ε C), c Encpk(m)

• Dec(sk, c) decrypts a ciphertexts with the secret key (sk) and recovers message (m) as the output, m 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 Evalevk(evk, C, c1, c2, ...cn).

Observation and Results:

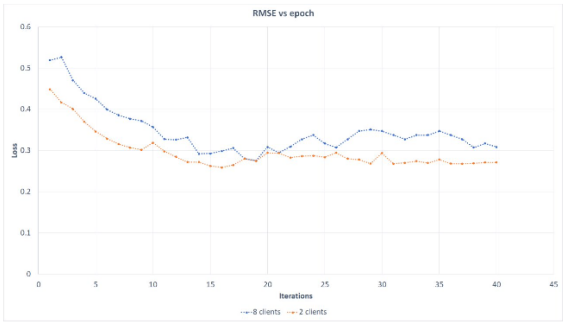

Fig. 6. RMSE values variation with number of Iterations in Homographic Encryption for two and eight clients.

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 the 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 a simple neural network which causes some amount of delay or latency in each iteration w.r.t the neural network without encryption.

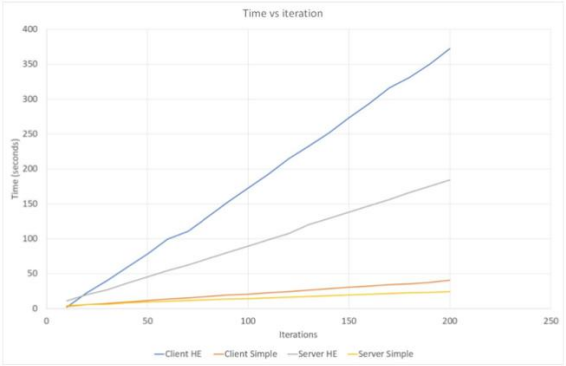

Fig. 7. Wall-clock running time for the client and server with and without Homographic Encryption in each iteration.

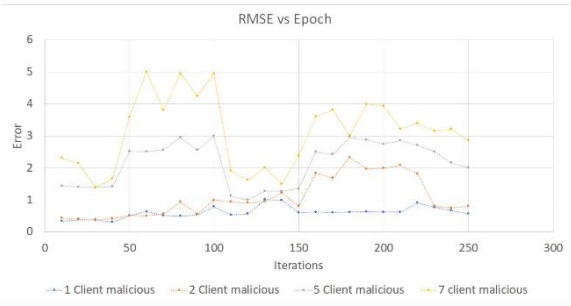

Fig. 8. RMSE values variation with number of Iterations for malicious clients

Because Homographic Encryption only works in a Semi-honest(curious to know) environment where all clients and servers want to know one another's parameters but can't manipulate them, we can see a number of bumps and changes in RMSE values in **Figure 8**. And in some cases, it isn't conceivable. So, we'll talk about a technique called Secret Sharing, which can address all of the above difficulties to some level.

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

Federated learning (FL), as discussed in section 3, is a new paradigm for distributed training of large-scale deep neural networks in which participants' data is kept on their own devices and only model updates are shared with a central server. The distributed nature of FL, on the other hand, creates new threats from potentially malicious participants. We'll look at the secret sharing technique to solve this problem. Secret sharing is an old cryptographic primitive with real-world applications in Bitcoin signatures and password management, for example. Secret sharing, on the other hand, has strong ties to secure computation and can be used for private machine learning, for example.

The essence of the primitive is that a dealer wants to split a secret into several shares and distribute them to shareholders in such a way that each shareholder learns nothing about the secret, but the secret can be reconstructed if enough people re-combine their shares. Intuitively, the issue of trust shifts from a single individual's integrity to the non-collaboration of multiple parties: it becomes distributed.

Secret Sharing's Key Steps: The main goal of this algorithm is to divide a secret into several distinct parts that must be encrypted.

• Let's call the secret we want to encrypt S.

• It's broken down into N parts: S1, S2, S3,..., Sn.

• After dividing it, the user chooses a number K to decrypt the parts and discover the original secret.

• It's chosen in such a way that we won't be able to find the secret S if we only know K parts (i.e., the secret S can't be reconstructed with (K – 1) parts or fewer.

• We can easily compute/reconstruct our secret code S if we know K or more parts from S1, S2, S3,..., Sn. This is referred to as the (K, N) threshold scheme.

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

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

The idea is to construct a polynomial of degree (K – 1) in which 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 by this polynomial using Lagrange's Basis Polynomial.

Let S = 65, N = 4, and K = 2 be the secret code.

• To encrypt the secret code, we first construct a polynomial of degree (K – 1).

• As a result, the polynomial should be y = a + bx. Our secret code is represented by the constant part 'a'.

• Assume b is a random number, such as 15.

• As a result, we get N = 4 points from this polynomial y = 65 + 15x.

Let's call those four points (1, 80), (2, 95), and (3, 110). (4, 125). Clearly, we can generate the initial polynomial from any two of these four points, and the constant term an in the resulting polynomial is the secret code.

• The Lagrange basis is used to reconstruct the given polynomial back. The term polynomial is used.

The Lagrange polynomial[12] is based on the idea of first forming the Lagrange identities, then summing these identities to get the required function that we need to find from the given points.

Overview of iterations in the secret sharing protocol:-

• At first, each client generates their own share of parameter shares.

• These shares are then distributed to all clients in such a way that each client owns the shares of the others.

• The parameters are sent to the server for aggregation after client distribution, and then the averaged parameters are broadcasted to all clients.

Observations in the secret sharing protocol:

As shown in Figure 9, the RMSE curves show a lot of variation when compared to Figure 7, and they are not converging or reaching global optima. It occurs because the precision value of the shared parameters, which is 3 in the case of secret sharing, and weights change so quickly in comparison to a simple neural network, resulting in high and low RMSE at times. And if we consider high precision, the time complexity increases because there are many steps, such as generating shares, distributing shares, model aggregating, and reconstructing shares using Lagrange interpolation, and it requires a high computational power CPU, indicating a trade-off between accuracy and resources required.

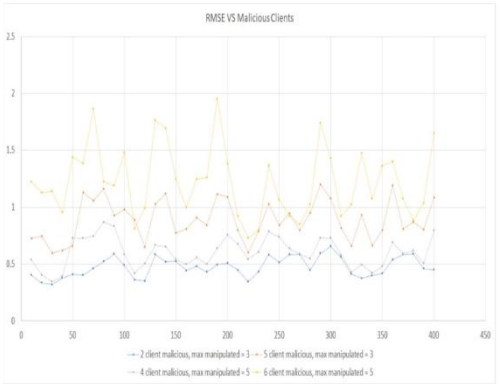

Fig. 10. RMSE values variation with number of Iterations for malicious clients

When the number of malicious clients and the number of shares, max, increase, as shown in figure 10, the RMSE values curves suddenly bump or increase as compared to figure 1. Failures are kept constant, and the maximum number of shares that can be manipulated is N-K shares, implying that we need at least K shares to reconstruct the secret, where N is the number of shares generated and K is the number of shares required to decrypt the

secret. And we can do an aggregation successfully if we have at least k shares on the server.

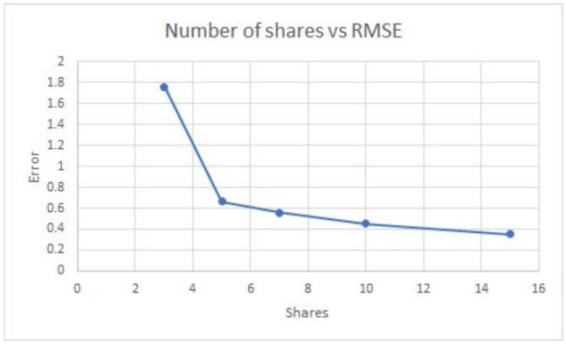

Fig. 11. RMSE variation with number of shares used

As shown in Figure 11, RMSE values decrease as the number of shares increases, which is understandable because each client will have a good mix of shares in this case, and malicious clients will not have a significant impact on the model. However, as the number of shares grows, the computation time grows, and we need more powerful resources to protect our model from malicious clients.

C. Comparison between the three protocols:

RMSE Comparison:

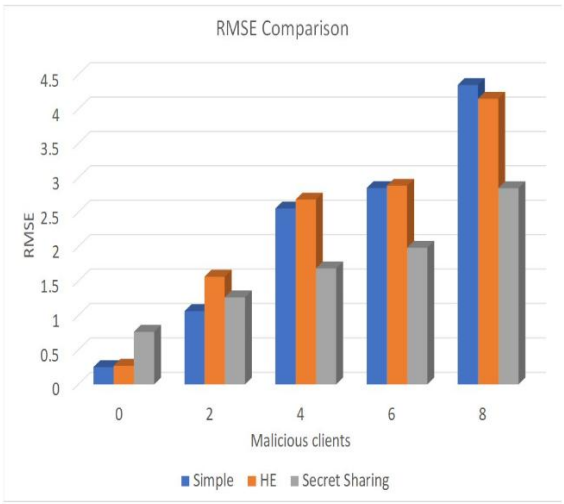

Fig. 12. RMSE comparison for all 3 protocols

Figure 12 depicts this. The Simple neural network and Homographic Encryption model performs better than the secret sharing technique initially with 0 or 2 malicious clients, but it performs worse as the number of malicious clients grows, indicating that secret sharing is better in a scenario with a large number of malicious clients. It is dependent on the state of the environment in order to determine which model to use.

Time Comparison:

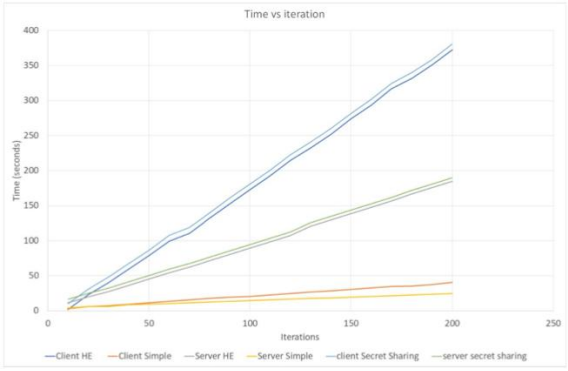

Fig. 13. Wall-clock running time comparison for all 3 protocols

Data-overhead Size Comparison:

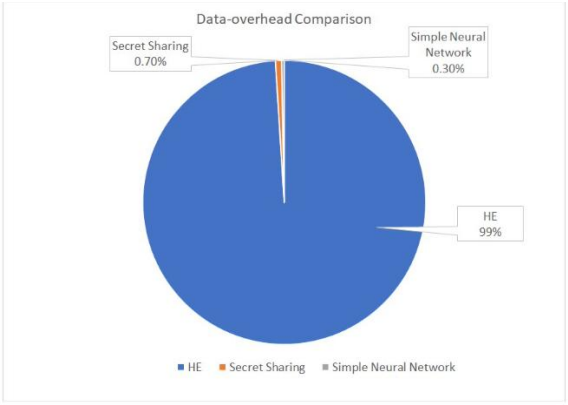

Fig. 14. Data-overhead comparison for all 3 protocols

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 a simple neural network is small which shows the trade-off between the privacy and computation time.
As shown in **figure 14**, the size of data overhead is too much which is almost(100%) for Homographic Encryption as compared to the other two. Each parameter is encrypted and encoded which has approximately 20000000 bytes(20 MB) length. On the other hand, the data overhead size is almost negligible in case of secret sharing and simple neural networks.

IV. CONCLUSION

We used different cryptographic algorithms, such as Homographic Encryption and Secret Sharing, to develop a privacy-preserving federated learning scheme in this report. We also discussed privacy issues and threats to the model in a general federated learning approach, and we attempted to use 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. After that, we created a set of secure protocols to implement the distributed Neural network linear regression model, using the secure aggregation scheme that we designed. Extensive tests were carried out to assess the efficacy and efficiency of both methods. The results of the experiments showed that using Secret Sharing allowed the neural network to be trained in a malicious and manipulated environment with some performance loss, and that secure aggregation computation and communication costs were reduced. However, Homographic encryption and the general approach were unable to handle user dropouts and manipulated users, despite producing some good and accurate performance results, and the data-overhead of communication in the case of Homographic Encryption was too large.

V. FUTURE SCOPE

We looked at both cryptography algorithms in which secret sharing was demonstrating some promising privacy results. However, we must improve the system's performance and attempt to reduce loss and time complexity in the following ways: -

1) Various methods for reassembling the secret, such as the Fast Fourier Transform and Newton's method, which can reduce time complexity and thus improve precision.

2) Distributed system scaling and design for a large number of clients

3) Create a system that combines Homographic Encryption and Secret Sharing.

4) By removing central authority, you can get to serverless architecture.

5) Using multi-threading to check the beats of all the instances

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 Lud- wig"HybridAlpha: An Efficient Approach for Privacy-Preserving Feder- ated 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]. QinbinLi,ZeyiWen,ZhaominWu,SixuHu,NaiboWang,BingshengHe, "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]" *Universite´ 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]. YangLiuZhuoMa, XimengLiu, SiqiMa, SuryaNepal, Robert.HDeng, 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 SecurityWiki"https://doubleoctopus.com/security-wiki/encryption-and-cryptography/secret-sharing/

[12]. Wikipedia the free encyclopedia, "Lagrange polynomial"https://en.wikipedia.org/wiki/Lagrangepolynomial

[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 Dro¨ge, Michael Moeller, "Inverting Gradients – How easy is it to break privacy in federated learning?" *arXiv:2003.14053 [cs.CV]*