Homomorphic Encryption on Medical Data: A Survey Study to Ensure Privacy and Operational Efficiency

Authors: Blaine Steck, Morgan Rogers, Asim Javed, Mohamad Abboud, Evan Jenkins, Michael Nweke

School of Computing and Engineering: University of MIssouri - Kansas City

CS5596A Computer Security I: Cryptography

Sravya Chirandas

06Nov2022

Abstract

We see competent, deeply learning models in medicinal applications or financial services that never reach the end user (Vepakomma et al., 2019). The proof of concept these models are trained on synthetic data that is openly available, however, in order to work in the real world these models need actual current data of the patients as the patterns in the data change over time which requires models to relearn (also known as incremental learning). The main problem lies in the fact that hospitals are not allowed to share data records of patients, such as their age, height, number of hospital visits in a month, heart rate, or their insurance company. This is due to data privacy laws such as HIPAA (Health Insurance Portability and Accountability Act) which protects individually identifiable health information, which bars hospitals from sharing patient data (Humphrey & advisor, 2021).

There is a risk of data leakage during the machine learning process; the attackers or hijackers can access the client's private data, and hence the service users' privacy is at stake. To deal with this issue, collaborative learning was proposed such as split learning, where a machine learning model is split into two (as in the name “split”) (Poirot et al., 2019). The computationally inexpensive layers, such as the input layer and one hidden layer (intermediate layer) trains at the client end. The model weights and activations are transferred through secure channels to the server when training. It trains the rest of the model it has and after completing forward propagation, sends it back to the client to complete backpropagation. This process goes on until the model

has been trained thoroughly. During this process, the server could never directly access the client's raw data.

Two major problems with modern machine learning models arise in models such as these. One issue is the server space required for large scale machine learning. Many organizations involved in larger scale machine learning make use of cloud computing services to handle large portions of their computing needs (Pop, 2016). The solution of cloud computing, however, causes the second problem: that of security. The simple solution to the problem of security of data stored on the cloud server is the encryption of data. The difficulty of encrypting machine learning data is that the model weights must be constantly modified as the model trains. Traditional encryption methods (such as AES) require that data be decrypted every time any modification needs to be made. The need to decrypt/re-encrypt data constantly can prove to be computationally expensive in the case of machine learning training. Fortunately, newer encryption schemes propose methods in which data can be modified without the need to be decrypted first. In this paper, we will discuss a category of encryption schemes known as fully homomorphic encryption. We will make use of a typical set of machine learning weights, and several different fully homomorphic encryption (FHE) schemes to help determine which scheme(s) are most useful when utilizing FHE schemes on such data.

Homomorphic encryption is still a relatively new encryption paradigm, and as such it can be somewhat challenging to select the most appropriate scheme for a given application. While the most recently developed schemes are considered to be the fourth

generation of homomorphic encryption, there can often be cases in which third, second, or even first generation schemes are preferable. In this paper, we will survey python implementations of the CKKS homomorphic encryption scheme when used on typical machine learning weights. We will present analytics to help inform developers/server architects to select the most ideal homomorphic encryption scheme to suit their particular needs. Source code will also be provided regarding the practical tests we will be performing using our sample data over each encryption scheme.

Background

In a more traditional configuration, a machine learning model may store its data (referred to as “weights” in most of this paper) on the organization/company’s server. Modern machine learning and server configurations can make this impractical, however, as the popularity and availability of cloud server solutions is appealing. Many organizations may opt to use a cloud server in order to store the weights from their machine learning algorithms. This solution can mitigate the issue of scalability when it comes to server capacity, however in the case of machine learning, many models (especially those used in medical applications) make use of data that is protected under HIPAA law (Mooney et al. 2018). In the case of HIPAA, security and privacy is of utmost importance, so using secure encryption schemes to protect data security is the natural solution. For many applications like this, a configuration involving a user, server, and data analyst can be used. On the user end, the user inputs information (the data that will be used to train the model) and the data is encrypted before being sent to the

server. The server (in this case a cloud solution) stores the data until it is requested by the data analyst, and the data analyst uses their private key to decrypt the data for analysis. This raises one major problem: How to handle updating the encrypted data on the server? Using more traditional encryption schemes, two “practical” solutions could be employed. One solution is to have the data analyst end of the configuration take on the task of decrypting and modifying all data. This is not ideal, as it would require much more data transfer than should be needed, as data would need to be sent from server to the data analyst, before being sent back to the server again. This quickly begins to seem untenable, as proper machine learning algorithms require frequent updates to their core data weights. The other solution is to provide a decryption key for the server to use. This allows for the server itself to perform any updates to the weights as needed, however presents a notable security flaw; that being the raw encrypted data being stored in the same place as the key needed to decrypt it. It is clear a solution needs to be implemented that meets both the practicality of allowing the cloud server to perform operations on the data while avoiding the security pitfalls traditional encryption techniques can cause when used in such a configuration.

Fortunately, a solution to the above presented issues has previously been proposed. The use of fully homomorphic encryption (FHE) schemes can allow for operations to be performed on encrypted data, without any need to perform any decryption first (Wood et al. 2020). When using FHE in the above proposed configuration: The user would input their training data, which is then encrypted using an encryption key (usually the public key in the case of schemes using an asymmetrical

encryption scheme). This encrypted data is then sent to the server. The server is provided only with the information needed to perform the operations on the encrypted data. In this way, the server is “blind” to the actual content of the data, but still able to accurately update the information contained within. The data analyst will decrypt data sent to them by the server locally using their decryption key.

The basis of homomorphic encryption, is that given plaintext (P_1 and P_2) and ciphertext (C_1 and C_2) where each ciphertext has been encrypted using the same encryption key, an operation such as $C_1 + C_2$ will decrypt to the same value as $P_1 + P_2$ (Yi et al. 2014). In the case of a fully homomorphic scheme, it is possible to perform both addition and multiplication operations on the ciphertext. Some more traditional encryption schemes, such as unpadded RSA are “partially” homomorphic. However the difference between partially homomorphic and fully homomorphic schemes is that non-fully homomorphic schemes can only perform one of the two operations (that is to say, depending on the scheme, they can perform addition *or* multiplication, but not both) which makes these schemes less than practical for most uses when performing many operations on the encrypted data is valued (Yi et al. 2014).

The encryption scheme we will be analyzing in this study is known as CKKS . CKKS: The Cheon-Kim-Kim-Song scheme performs computations on vectors and complex values. A message which is a vector of values is encoded into a plaintext polynomial and then encrypted using a public key. Once the message is encrypted into a set of polynomials, CKKS offers a couple of functions that can be performed on it such

as addition, multiplication, and rotation. In order to decrypt, a secret key will be used on the set of polynomials which will return an encoded message which after decoding will reveal the message (Huyhn D. 2021). CKKS is the most modern of available homomorphic encryption schemes, and as such it is considered the most efficient for most purposes. Our focus will be finding which implementations of this scheme are most ideal for practical use.

Methodology

In this study, we will utilize open source python libraries Ibarrond/Pyfhel (Ibarrond) and TenSEAL (Benaissa et al. 2021) (others to be added after initial testing). These libraries contain implementations of the CKKS encryption scheme. These schemes will be used to encrypt a randomly generated set of machine learning weights. These weights are contained in a 4 dimensional num.py array of float values. The array itself is considerably large (thousands of entries in the array) so for practicality of testing, we will only encrypt a subset of the values. Our program will record analytics to an output file. These analytics will include encrypted file size (relative to plaintext), speed of encryption of data, speed of decryption of data, speed of modification of data (that is, performing homomorphic operations), and decrypted data accuracy.

Encrypted file size is an important metric to measure, as the proposed solution employing these FHE schemes relies on the usage of cloud servers. Larger capacity of cloud servers can become costly, and excess use of storage space should be avoided

when possible. File size must be checked, as unlike traditional encryption schemes, FHE schemes (particularly schemes with higher security level) often do not necessarily have a 1:1 ratio of plaintext size to ciphertext size after several homomorphic operations. Most homomorphic schemes have methods built in to reduce this effect, however it's important to the overall effectiveness of these measures (Coron et al. 2011). The ciphertext size is often larger than the original plaintext; a difference that can be exacerbated after a large number of homomorphic operations have been performed on the ciphertext.

Since the encryption of the original data will typically be performed on the user end, it is important to determine relative encryption speed. Users may be less likely to use a system if speeds are slow, therefore it is in the best interest of organizations looking to utilize an encryption scheme as described to ensure that their chosen method does not cause unnecessary slowdowns on the user end. This is particularly true if large amounts of data are expected to be encrypted on the user end prior to being sent to the server. The different schemes will have their encryption speed compared relative to one another, in order to help account for individual differences in hardware setups as it is difficult to predict specific hardware configurations on the user end, which can cause significant differences in the real-time encryption speed.

Speed of decryption is also an important metric to track. If the organization's machine learning solution requires large amounts of information to be sent to the data analyst frequently, decryption speed will be key in choosing the proper encryption

scheme as decryption will occur on the data analyst end of the service configuration. The different encryption schemes will be compared relative to one another once again, as differences in hardware configuration on the data analyst end can cause notable differences in real-time decryption speed.

The speed of the homomorphic operations may be one of the more critical metrics to track, as any given machine learning solution will require a large number of modifications to the data encrypted on the cloud server. After all, the main problem with machine learning that necessitated the use of homomorphically encrypting data was the large number of operations that would need to be performed as the machine learning model is trained. We will perform a large number of homomorphic operations on the ciphertext and compare the rate each scheme is able to perform them relative to one another.

Decrypted data accuracy is also measured, as most FHE schemes do cause some level of “noise” in data that has been encrypted and modified while encrypted. This is due to the fact that many schemes utilize some “approximate” calculations when performing multiplication or addition over ciphertexts (Fan et al. 2012). Different FHE schemes have different solutions for dealing with errors in the data. In some schemes, a limited number of homomorphic operations can be performed before the ciphertext needs to be “refreshed” using a process known as “bootstrapping” (Gentry C. 2009). Some of these solutions can be more computationally expensive than others, and if a machine learning solution requires that both a significantly large number of

homomorphic operations occur on encrypted data while ensuring accuracy, this metric becomes critical to examine. In order to test data accuracy between encryption libraries, we will perform operations on the plaintext of the data, and homomorphically perform the same operations on the ciphertext. The resulting decrypted ciphertext will be compared to the plaintext that was never encrypted for any data errors. The average deviation will be recorded and used in evaluating the various encryption libraries.

Results

Encrypting 1500 data arrays from a machine learning model

Python library	Encrypt time (ms)	Decrypt time (ms)	Time of operations (ms)	Average deviation
Pyfhel	28768.125	9450.262	6349.7080802 9174	1.427E-04
TenSEAL	5588.278	1110.527	1004.612	2.299E-06

Our initial results show a significant speed difference between the Pyfhel and TenSEAL libraries. Running on the same hardware, TenSEAL was able to encrypt 1500 arrays over 5x faster than Pyfhel, and decrypt nearly 9x faster. TenSEAL was also able to perform homomorphic operations around 6x faster than Pyfhel. When taking data accuracy into account, TenSEAL also showed a much smaller variance in data. While

both of these libraries make use of the CKKS encryption scheme, it seems clear TenSEAL has a much more efficient implementation.

Clearly, only two python libraries aren't enough of a comparison to definitively say which implementation should be used. However between the two used in this initial analysis TenSEAL appears to be a more promising option for practical purposes. We intend to perform the same analysis on other CKKS implementations for a more complete survey. We also intend to analyze encrypted file size in order to determine if any of these implementations struggle with bloating file size when creating/modifying ciphertexts.

Another notable factor is that the performance analysis above only shows performance across a single thread. It is possible that these implementations function better when multi-threaded, which would imply an improvement in performance in the more realistic setting of running these algorithms in a cloud computing configuration.

Both Pyfhel and TenSEAL have multiple other variables to modify the encryption scheme, leading to tradeoffs between higher security and lower efficiency (or vice versa). It is possible through further testing we will find that these different implementations handle higher security factors more efficiently than the others. This may be a factor we consider in future iterations of this paper, or a suggestion for further research based off of this survey.

In conclusion, our baseline results show that we can quantitatively compare these different implementations in a way that should be simple to understand for those interested in potentially using them for the purpose of encrypting data (and in particular machine learning weights) in a way that they can be homomorphically operated upon.

We will continue to revisit these results and add to them as new implementations are tested and other factors to include in our analysis emerge.

References

Al Badawi, A., Bates, J., Bergamaschi, F., Cousins, D. B., Erabelli, S., Genise, N., ... & Zucca, V. (2022). OpenFHE: Open-source fully homomorphic encryption library. *Cryptology ePrint Archive*.

Benaissa, A., Retiat, B., Cebere, B., & Belfedhal, A. E. (2021). TenSEAL: A library for encrypted tensor operations using homomorphic encryption. *arXiv preprint arXiv:2104.03152*.

Chen, C., Zhou, J., Wang, L., Wu, X., Fang, W., Tan, J., Wang, L., Liu, A., Wang, H., Hong, C. (2020). *When Homomorphic Encryption Marries Secret Sharing: Secure Large-Scale Sparse Logistic Regression and Applications in Risk Control.* https://arxiv.org/abs/2008.08753

Chillotti, I., Gama, N., Georgieva, M., & Izabachene, M. (2016, December). Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *international conference on the theory and application of cryptology and information security* (pp. 3-33). Springer, Berlin, Heidelberg

Coron, J. S., Mandal, A., Naccache, D., & Tibouchi, M. (2011, August). Fully homomorphic encryption over the integers with shorter public keys. In *Annual Cryptology Conference* (pp. 487-504). Springer, Berlin, Heidelberg.

Damkliang, K. (2019, December 17). *AT&T database of faces*. Kaggle. Retrieved September 11, 2022, from https://www.kaggle.com/datasets/kasikrit/att-database-of-faces

Ducas, L., & Micciancio, D. F. (2015). Bootstrapping Homomorphic Encryption in Less Than a Second.

Dwork, C., Kenthapadi, K., McSherry, F., Mironov, I., & Naor, M. (2006). Our data, ourselves: Privacy via distributed noise generation. *Advances in Cryptology EUROCRYPT 2006*, 486–503. https://doi.org/10.1007/11761679_29

Fan, J., & Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*.

Fang, H., & Qian, Q. (2021). Privacy preserving machine learning with homomorphic encryption and federated learning. Future Internet, 13(4), 94. https://doi.org/10.3390/fi13040094

Gentry, C. (2009, May). Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing* (pp. 169-178).

Humphrey, B. A., & advisor, K. P. D. S. (2021). *Data Privacy vs. innovation: A quantitative analysis of artificial intelligence in healthcare and its impact on Hipaa regarding the privacy and security of Protected Health Information/ by* (dissertation). ProQuest Dissertations Publishing, Ann Arbor, MI.

Huynh, D. (2021, March 15). *CKKS explained: Part 1, Vanilla Encoding and decoding*. OpenMined Blog. Retrieved November 5, 2022, from https://blog.openmined.org/ckks-explained-part-1-simple-encoding-and-decoding/

Ibarrond. (n.d.). *Ibarrond/pyfhel: Python for homomorphic encryption libraries, perform encrypted computations such as sum, mult, scalar product or matrix multiplication in python, with numpy compatibility. uses seal/palisade as backends, implemented using Cython.* GitHub. Retrieved November 5, 2022, from https://github.com/ibarrond/Pyfhel

Introduction to the BGV encryption scheme. Inferati. (2022). Retrieved November 5, 2022, from https://www.inferati.com/blog/fhe-schemes-bgv

Introduction to the BFV encryption scheme. Inferati. (2021). Retrieved November 5, 2022, from https://www.inferati.com/blog/fhe-schemes-bfv

Kumari, K. A., Sharma, A., Chakraborty, C., & Ananyaa, M. (2022). Preserving Health Care Data Security and privacy using Carmichael's theorem-based homomorphic encryption and modified enhanced homomorphic encryption schemes in Edge Computing Systems. Big Data, 10(1), 1–17. https://doi.org/10.1089/big.2021.0012

Mooney, S. J., & Pejaver, V. (2018). Big data in public health: terminology, machine learning, and privacy. *Annual review of public health*, *39*, 95.

Poirot, M. G., Vepakomma, P., Chang, K., Kalpathy-Cramer, J., Gupta, R., & Raskar, R. (2019, December 27). *Split learning for collaborative deep learning in Healthcare*. arXiv.org. Retrieved September 11, 2022, from https://arxiv.org/abs/1912.12115

Pop, D. (2016). Machine learning and cloud computing: Survey of distributed and saas solutions. *arXiv preprint arXiv:1603.08767*.

Vepakomma, P., Gupta, O., Swedish, T., & Raskar, R. (2018, December 3). *Split learning for health: Distributed deep learning without sharing raw patient data*. Retrieved September 11, 2022, from https://arxiv.org/abs/1812.00564

Vepakomma, P., Singh, A., Gupta, O., & Raskar, R. (2020). Nopeek: Information leakage reduction to share activations in distributed deep learning. *2020 International Conference on Data Mining Workshops (ICDMW)*. https://doi.org/10.1109/icdmw51313.2020.00134

Wood, A., Najarian, K., & Kahrobaei, D. (2020). Homomorphic encryption for machine learning in medicine and bioinformatics. *ACM Computing Surveys (CSUR)*, *53*(4), 1-35.

Yi, X., Paulet, R., & Bertino, E. (2014). Homomorphic encryption. In *Homomorphic encryption and applications* (pp. 27-46). Springer, Cham.