# Homomorphic Encryption on Network traffic data for Neural Networks

Nam Khanh Cao *24577607* Diyon Ratnayaka *14465417* Kim Thai Bao Huynh *14344358* Brendan Budniak *24976437* Aurora Wu *24631018*

***Abstract*—In the field of cybersecurity, encryption with neural networks to implement AI technology to ensure safety of data from vulnerabilities. The significance of the proposed research lies in its necessity for safely securing network data and conducting mathematical operations on that data without compromising the encryption. Given the rising inventions of AI technology in various domains, it is important for researchers to incorporate studies in neural network cryptography to address the current challenges. The goal and objective of this project is to provide a more accurate, secure cryptography solution than traditional methods.**

***Index Terms*—Cryptography, Neural Network, Homomorphism, Encryption technique, Data, Network traffic**

## Introduction

AI technologies and machine learning algorithms have achieved mainstream adoption across numerous fields, from small businesses to large enterprises, and data privacy is an issue impacting individuals in society that requires significant re-evaluation to keep up with the demands on how businesses and individuals interact. AI technology provides convenience to both organisations and individuals within society. However, the omnipresence of technology in society can lead to overlooking the underlying negative implications and security concerns. The Paillier cryptosystem is a partially-homomorphic encryption technique that maintains the privacy of the user's original data whilst allowing mathematical operations on encrypted data. Its unique homomorphic properties ensure complete confidentiality when supplied to machine learning algorithms. This property contains significance in privacy-preserving computation, especially in multiparty computation and privacy-preserving machine learning contexts. The goal is to utilise the homomorphic properties of the Paillier cryptosystem to improve machine learning algorithms, including K-nearest neighbour, support vector machine, random forest, and decision tree. The KDD Cup 1999 dataset is selected as the source for training and evaluating the machine learning model. We provide detailed steps with code snippets and guidelines on how we achieved this through the Python programmed Paillier encryption scheme.

The remaining sections of the paper are structured as follows: Related Work provides a comprehensive overview of related work. Problem formulation for provides current issues with current usage of encryption,in machine learning tasks. Proposed Solution provides solutions to the problems listed in the Problem Formulation section, while Experiments provide proof of concept. Conclusion concludes what was learnt from the research, and References section show references used in this paper.

## Related Work

### A. Background

In 1999, Pascal Paillier proposed the Paillier cryptosystem in his publication 'Public-Key Cryptosystems Based on Composite Degree Residuosity Classes,' an encryption scheme known for its additive homomorphism properties. Paillier introduced the theoretical framework, including the mathematical operations and properties that provide the foundation for the security of the encryption technique. Unlike traditional encryption methods, Paillier encryption allows the addition of ciphertext that corresponds to the sum of the plaintexts. For further details of the Paillier cryptosystem's theoretical framework, we refer readers to [1]. There is a growing interest among researchers and developers in implementing and leveraging the homomorphic encryption in various domains, such as cloud-based machine learning solutions used in medical and financial fields that are constantly dealing with sensitive data. The paper 'CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy' [12] discusses the concept that combines homomorphic encryption techniques with neural networks. The core idea is to remain encrypted throughout the entire machine learning process, including training phases, to address concerns regarding privacy and security in machine learning applications. The author discusses one of the challenges and limitations of CryptoNets: the computational overhead with homomorphic encryption as it adds extra computational complexity.

### B. Related Research

In 2022, Wingarz et al. introduced a novel approach to privacy-preserving Convolutional Neural Networks (CNNs) using homomorphic encryption [10]. The network, implemented via SEAL-Python, a Python wrapper, achieved a precision in training that led to a final scaling of encryption

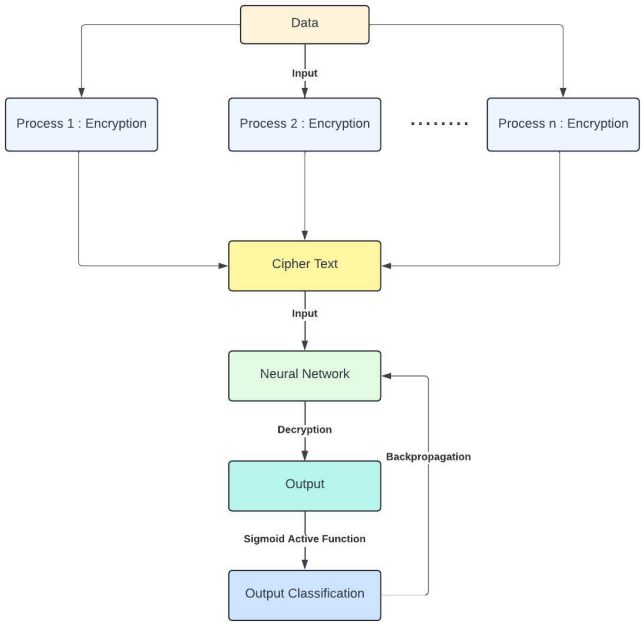


Fig. 1. Process of training

numbers up to $2^234$. However, the resource consumption of the encrypted network is still significantly higher than the baseline system, 'we can approximate the size of a single ciphertext by $2 * Nlog2(q)$ bits' as stated by the author. In this project, we will combine machine learning algorithms with Paillier's cryptosystem to enable the program to handle large network traffic datasets effectively. Finally, we will discuss the challenges such as consumption and its suitability for real-life applications in different domains.

## Problem Formulaion

Securing network traffic data is a significant concern in the field of digital networking. Machine learning algorithms are being used to analyze and make decisions based on this data, which is crucial for improving network speed and ensuring strong security. Nevertheless, a major security vulnerability arises during the data analysis stage. Traditional encryption techniques, however effective in securing data when it is stored and being sent, are insufficient for data processing [18]. Especially in machine learning applications, algorithms need access to specific, unencrypted data for training and predictive analytics, which presents a significant privacy threat [17]. Decrypting sensitive information, vital for useful analysis, exposes it to potential security breaches. The situation highlights the immediate requirement for advanced encryption methods that can secure data at all stages, even during machine learning procedures, to comply with strict global data protection rules. The main challenge in implementing machine learning for network traffic analysis is the requirement to decrypt data, which significantly compromises security. In order to train successfully and generate correct predictions, machine learning algorithms need access to comprehensive, frequently unencrypted data. This method raises significant security vulnerabilities, especially when private or sensitive information is implicated.

Constraints of Existing Solutions include :

### C. Security vulnerabilities

Current solutions decrypt data during processing, making it vulnerable to potential cyber threats and unauthorized access [13]. These solutions leads to significant security vulnerabilities, particularly when performing inside strict data protection standards.

In cryptography, encryption keys are often required to be 256 bits in length. Mathematically, this can be expressed as:

$$K = \{k \in \{0,1\}^{256}\}$$

This notation means that the set $K$ consists of all binary strings of length 256. An encryption key $k$ must satisfy the condition:

$$k \in K$$

Thus, each key $k$ is a 256-bit binary string. Although, this is fairly large number ($2^{256}$), it is crucial to consider the harder it is to compute the key pairs, the better the security.

### D. Data encoding

The mismatch between the data types used in machine learning (often real numbers) and those required for secure encryption (typically integers) presents another major challenge [14]. This mismatch in encoding is a challenge when attempting to directly use encrypted data in machine learning models, typically requiring supplementary preprocessing procedures that may introduce inaccuracies or decrease model precision.

$$\mathbb{R} \neq \mathbb{Z} \tag{1}$$

### E. Computational overhead

The computational requirements for handling encrypted data are significantly more than for unencrypted data [16]. This increase in computational load often results in slower processing speeds, making real-time analysis and decision-making more challenging.

These issues highlight the importance of a robust solution that can securely manage encrypted data in machine-learning processes without compromising performance, accuracy, or speed. So, our proposed solution's main aim is to address these issues by enhancing security measures, boosting computing performance, and directly resolving data encoding mismatches. This would result in a more robust and efficient framework for processing network traffic data.

The crucial desire for homomorphic encryption arises from its ability to execute calculations on encrypted data without decryption, effectively tackling several critical issues as previously defined. Traditional encryption methods preserve data when it is stored and sent, but they are not effective in

securing sensitive information during processing, which leaves it vulnerable to potential security breaches. Homomorphic encryption ensures that data remains encrypted during the analysis process, which not only preserves the confidentiality and integrity of the data but also helps ensure compliance with strict privacy regulations [19]. This method tackles the issue of data exposure during processing, allowing for secure and confidential calculations that are crucial for handling sensitive network traffic data in the fields of machine learning and data analytics. The primary goal of using Paillier encryption in our research is to examine and showcase its feasibility and effectiveness in handling encrypted network traffic data through the use of machine learning methods. The specific objectives include evaluating the performance implications of encryption on processing speeds and algorithm accuracy, examining the computational overhead, and identifying the scalability in existing systems. Our intention is to prove the feasibility of Paillier encryption, not only as a theoretical model, but also as a realistic solution for real-time, secure data analytics, by defining these specific goals.

## PROPOSED SOLUTIONS

### F. Paillier Scheme

The primary focus of our research is homomorphic encryption, with a specific emphasis on the Paillier cryptosystem. Homomorphic encryption allows for addition and multiplication operations on ciphertext, meaning that computational tasks can be performed on encrypted data without the need for decryption. This ensures that third parties, such as neural networks in our study, cannot access plaintext or extract useful information from the encrypted data. Paillier's cipher offers partial-homomorphic encryption properties due to the multiplicative property of the ciphertexts [1]. In the presented encryption scheme, firstly 2 prime numbers, p and q, are picked as a base to derive the public and private key from, the length of these prime numbers will determine the bit length of the public and private keys, and thus the limits of which overflows that can heavily corrupt encrypted data will occur [1].

Let $p$ and $q$ be distinct prime numbers of $N$ bits, denoted as $p, q \in \mathbb{P}_N$, where $\mathbb{P}_N$ represents the set of prime numbers with $N$ bits.

These keys go through some modulus operations to form a group of values within the modulus of n squared, where n is the product of p and q.

$$n = p \cdot q \tag{2}$$

Equation (2) is for determining the length of the key, this sets a precedent for scalars to be within a certain range, and with encoded numbers, we need keys with at least 256-bits without consistent corruption of data as encoded numbers take up significant amount of memory in order to keep floating point precision. We then generate g where g is random value of the multiplicative group of n squared. This g is a crucial part for as equation (5), where it is the modular multiplicative inverse using g as base. Let $r$ be a pseudo-random value that is uniformly distributed in the interval $[0, n)$, denoted as $r \in [0, n)$. Encryption is done through (n,g), thus the encryption process where m is the plaintext, c is the ciphertext is :

$$c = (g^m \cdot r^n) \bmod n^2 \tag{3}$$

We then compute the lambda, crucial for the decryption process as it is used as knowledge to the factors of n. We then compute the equation (5) from generating g where g is the multiplicative group of n squared.

$$\lambda = LCM(p-1, q-1) \tag{4}$$

We then compute the equation (5) from generator g and (4) to verify our last parameter for the decryption process to generate our private key:

$$\mu = (L(g^\lambda \bmod n^2)^{-}1 \bmod n \tag{5}$$

Where L is defined as :

$$L(x) = \frac{x-1}{n} \tag{6}$$

With this, we have our private key, we can now start decrypting with the equation :

$$m = L(c^\lambda \bmod n^2) \cdot \mu \bmod n \tag{7}$$

### G. Encoding floatingp point numbers

Due to working with modulus, the encryption algorithm requires the use of integers. It is then crucial to develop an encoding wrapper, to perform encryption without corrupting data. To decrypt, decoding must be done with the encoded number in order to retrieve the value.

This is done through extraction of several parts of the floating point number. According to the IEEE numeric standards, IEEE-754 describes double-precision floating point formats to be of 2 integral parts. A floating point number has a sign bit, a 11-bit exponent and a 52-bit mantissa, adding up to a total of 64 bits [2].

$$\overbrace{\text{Sign Bit}}^{\text{S}} \quad \overbrace{\text{Exponent (11 bits)}}^{\text{EEEEEEEE}} \quad \overbrace{\text{Mantissa (52 bits)}}^{\text{MMMMMMMMMMMMMMMMMMMM}}$$

Sign Bit (S): 0 for positive numbers, 1 for negative numbers.
Exponent (E): 11-bit field representing the biased exponent.
Mantissa (M) : 52-bit field representing the normalized fraction

First, we extract the stored base-2 exponent ($e$) and the mantissa ($m$) to then calculate the least significant bit (LSB), denoted as $LSB = \frac{1}{2^{\text{Log}_2 \text{ base}}}$. This helps derive the precise exponent from $e$ through the division of the $\text{Log}_2$base

The precision of the encoding is determined by these operations. The default base is 16 ($\text{Log}_2$ base $= 4$) for faster computations, although it leaks some information from the exponent, which is not alarming.

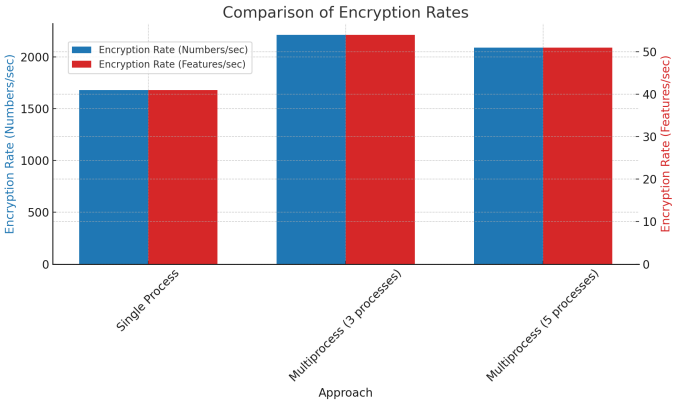


Fig. 2. Encryption Rates (Numbers/sec)

Let $x$ be the floating-point number. The integer representation of $x$ is obtained by rounding down the result of $x$ in fractions multiplied by the base to the power of the negative exponent:

$$\text{integer}(x) = \lfloor x \cdot \text{base}^{-e} \rfloor$$

Due to memory limitations within the encoding, encoded values can be expensive memory-wise. Thus, it is crucial to store the exponent ($e$) so that the value can be derived when multiplication between encoded numbers occurs.

### H. Methods

*1) Encryption:* We will be utilising python. Public/Private keys need to be of a certain bit length, thus for ease of use, python was used to implement the encryption scheme due to its arbitrary precision integers [4]. We created 3 objects responsible for the encryption, the paillier key generator, public and private key objects, to then encrypt any value we require as a wrapped Encrypted object. Paillier Key Generator object generates all attributes necessary through a multitude of utility functions implemented for prime generation, modular multiplication, modular multiplicative inverse, etc. It then uses these attributes to create a public key and private key object with the attributes stated above. For public key, it requires value $n$ from (2) and generator g, in addition to a random integer r from 0 to the n. The private key requires value $\mu$ from equation (5) and $\lambda$ from equation (4).

*2) Encoding Wrapper:* We also implemented an Encoding object to wrap the scalar into a suitable integer format, as floats and negative numbers will not work with the encryption, with the idea taken from data61, a github user [3]. We will implement operator overloading so that when Encrypted Object were to perform any operation such as multiplication, division, subtraction or addition, it will perform a specific operation to avoid type mismatch errors.

*3) Performance increase using numpy in training:* We then implemented 2 few basic Neural Network, Multi-layer Perceptron, and Recurrent Neural Network as an object. These neural networks require the use of lists, thus frameworks that utilise a specific data structure for their vectors such as numpy or tensorflow cannot be used. However, since the output has to be decrypted either way, we can save performance costs by decrypting and converting all to numpy arrays for better matrix multiplication instead of relying on list comprehension [4].

*4) Use of processes for encryption:* We implemented the use of processes for encrypting. Although it is not perfect, it can potentially reduce time taken for encryption by n processes. Using the multiprocessing module, we can run multiple processes from the CPU to encrypt, thus time complexity can be reduced by the division of n times. We experimented with threads, however, due to python being a single threaded language, we obtained the same result, making multiprocessing more practical in a language like python. Of course we require a limit on the amount of processes due to hardware requirements.

TABLE I
CLASSES OF ATTACKS FOR ML LABELING

| Class labels | Description |
|---|---|
| Normal Packets (normal) | Regular network traffic |
| User to Root Attacks (u2r) | Unauthorized privilege escalation |
| Denial of Service Attacks (dos) | Overwhelming a system |
| Remote to Local Attacks (r2l) | Gain local access attacks |
| Probing (prob) | Scanning for information |

## EXPERIMENT

**We are using google colab with no subscription for all experiments conducted.**

### I. Dataset and Preprocesing

The benchmark dataset chosen for is **KDD-Cup 1999** a dataset full of classified intrusions with information of corresponding packets as features [8]. We will be using KDD-Cup 1999 as our dataset, this is due to its feature diversity as well as its size. Although it has a variety of labelled attacks, we decided to group certain attacks into 5 main labels. First, we have normal packets, they are not attacks and should be classified by an intrusion detection model. We then user to root attacks, which is basically unauthorized privilege escalation [7]. We also have denial of service attacks, denying service for users by overwhelming a system. We also have remote to local, basically gaining local access attacks through backdoors [7]. The final class is probing, scanning and gaining information on the network [7].

### J. Encryption rate

For our experiment, we ran tests with a public/private key pair of **256 bit length**. This is done to avoid any overflow issues caused by the encoding. 128-bit length keys caused overflows frequently. We performed experiments then encryption rate with processes, first we ran single process as baseline to see if it is actually quicker to run multiple processes, then we ran 3 processes and then 5 processes.

From table II and III, you can see that use multiple processes does increase the encryption rate at a significant rate, however,

it also depends on hardware specifications, so CPUs with multiple cores might be able to efficiently run more processes [6]. Random access memory is also another bottleneck as processes can use all the memory allocated, leaving other processes performing encryption less room for to perform their operations [12].

### K. Neural Networks

We implemented 2 types of neural networks, one is the **Multi-layer Perceptron (MLP)** and the other is the **Recurrent Neural Network (RNN)**. The implementation is simple enough, however is able to demonstrate the practicality of the encryption.

For the neural networks, we implemented 2 layers as in terms of training performance, working with encrypted data is a big bottleneck due to the operations done with modulus of $n^2$. The minimum required key length for the implementation has to be 256-bits, as overflows easily occur if keys cannot keep up with encoded numbers. In order to perform training, we have to consider the lengths in which encoded numbers can be multiplied, thus it might be ideal to implement a layer by layer decryption then encryption process, in order to go through an activation function such as the sigmoid function, as a way to normalise the results to a certain range without sacrificing too much security. However, this comes with a huge performance dip to encrypt and decrypt again, adding several $O(n)$ functions is not ideal.

Weights for both neural networks are not encrypted, this also includes the hidden state used by the recurrent neural network. This is not because of an issue with the performance, it is due to a limitation of the Paillier-scheme, since it is a partially-homomorphic encryption scheme, it cannot perform multiplication operations between ciphertexts [1].

### L. Training

We established a baseline with unencrypted data, in order to compare results between performances of encrypted and unencrypted data. We ran the experiment with both neural networks at 5 epochs, taking measurements such as :

You can see hyperparameters specified within the table V, it will act as our controlled variable for this experiment. The parameters are set for ease of use when forwarding such that list comprehesion does not run into many index out of range issues. Thus, this focuses on extracting the performance of the encrypted data. Unencrypted data will be done with the same parameters to measure the difference between the time taken to train depending on whether the input data is encrypted or not.

TABLE II
COMPARISON OF ENCRYPTION RATES (NUMBERS/SEC)

| Approach | Processes | Encryption Rate (Numbers/sec) |
|---|---|---|
| Single Process | 1 | 1681 |
| Multiprocess (3 processes) | 3 | 2214 |
| Multiprocess (5 processes) | 5 | 2091 |

TABLE III
COMPARISON OF ENCRYPTION RATES (SAMPLES/SEC)

| Approach | Processes | Encryption Rate (samples/sec) |
|---|---|---|
| Single Process | 1 | 41 |
| Multiprocess (3 processes) | 3 | 54 |
| Multiprocess (5 processes) | 5 | 51 |

[a]Depends on hardware specifications.

Our evaluation criteria is shown from table IV with a description on which testing criterias we will be focusing on for our experiment.

### M. Results

*1) Initial Analysis:* For our first analysis, we succeeded in implementing the encryption algorithm, however only using a single process. Our results for the encryption rate had the same average as the result of using a single process in II and III. In addition, instead of utilising numpy arrays for backpropagation, we instead relied on list comprehension for our experiment, thus our training speed were significantly worse than our current implementation. The results in table IX were from a multi-layer perceptron neural network.

*2) Experiment results:* We performed tests for training for 10000 samples and testing for 2000 samples, higher than the initial analysis. However, accuracy seems to be higher due to the conversion into numpy arrays for faster matrix multiplication. For both the initial and current analysis of our training, you can see that high training and testing accuracy is still achievable for encrypted data, although the time taken to train has plenty to be desired. The increase in speed is a trade-off between security and performance, instead of decrypting after forwarding, we decrypt after a certain amount of layers to increase performance and to provide usage of an activation function.

*3) Comparison with base-line (unencrypted data):* As you can see from our results. Training time with unencrypted numbers is by far much quicker due to the lack of computational overhead caused by multiplication of encrypted and encoded

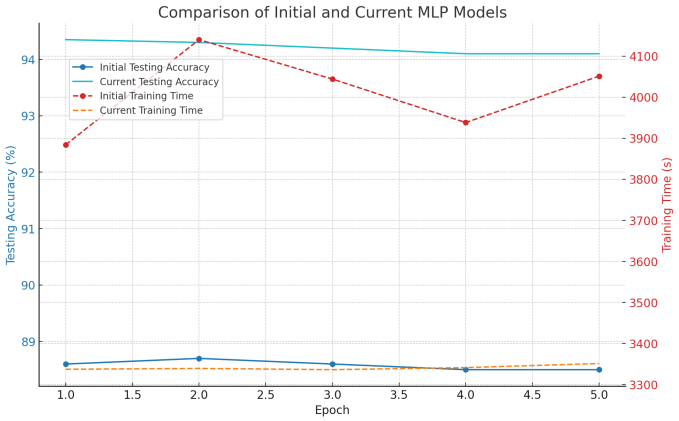


Fig. 3. Testing Accuracy (MLP

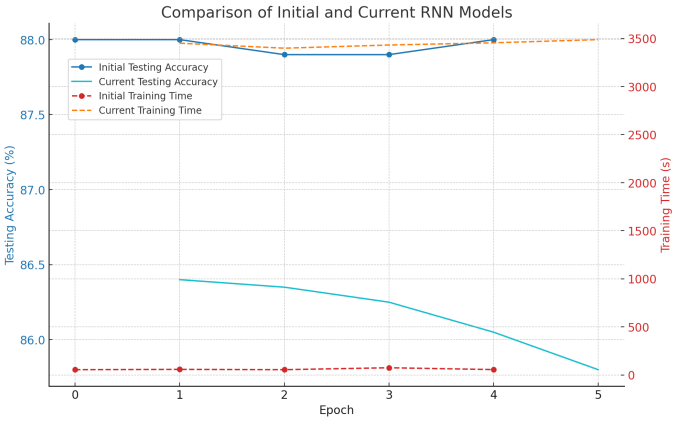


Fig. 4. Testing Accuracy (RNN)

numbers. Thus, in order to match the speed of training for unencrypted data, in addition to using them in large models that have millions of parameters, plenty of effort is still required.

## Conclusion

There is still progress to be done before homomorphic encryption to be applied to real Machine learning tasks. There are 2 main factors contributing to the success of homomorphic encryption. One is security, the length of keys play a detrimental role in securing the input data given. However, the higher the length of the keys, the more the performance of these machine learning algorithms suffer from the computational overheads caused by these issues, as you can see from this paper. As models continue to grow in size, the challenge continues to grow. There will be optimisations done, however, according to our research, there is always a trade-off between performance and security. The longer it takes for an attacker to compute the plaintext from the ciphertext, the higher the bit length of the encrypted numbers, the higher the computational costs from storing such numbers.

It is crucial to state that there are other schemes that provide homomorphic properties such as RSA [9], thus research is required to furthur investigate the practicality of all these schemes before giving a strong conclusion on the subject. There is room for growth within the idea of training during encryption, such as the usage of other languages other than python for our encryption and encoding, thus in the future, there will be analysis and implementations based upon languages that are not single threaded such as python for these ideas, as well as examinations on different encryption schemes within the field. Being able to use threads in processes create an advantage for encryptions to perform much quicker than multiprocessing [6], however, it is crucial that the system has enough memory to avoid mutex and I/O (input/output) locks [12].

TABLE IV
Evaluation Criteria

| **Criterion** | **Description** |
|---|---|
| Average Training Accuracy | Accuracy achieved during training on average |
| Average Training Loss | Loss incurred during training on average |
| Average Testing Accuracy | Accuracy of testing dataset on average |
| Time taken | Time taken for training (encrypted vs. unencrypted data) |

TABLE V
Hyperparameters

| **Criterion** | **Description** |
|---|---|
| Input Size | 41 |
| Hidden Size | 41 |
| Learning rate | 0.01 |
| Training Sample Size | 10000 |
| Testing Sample Size | 2000 |

TABLE VI
Initial Analysis (MLP)

| Epoch | Accuracy (%) | | Loss | Time |
|---|---|---|---|---|
| | Testing | Training | Training | Training time (hr) |
| 1 | 88.6 | 97.61 | 0.0193 | 1:04:44 |
| 2 | 88.7 | 98.51 | 0.0120 | 1:09:40 |
| 3 | 88.6 | 98.61 | 0.0108 | 1:04:39 |
| 4 | 88.5 | 98.69 | 0.0101 | 1:05:58 |
| 5 | 88.5 | 98.78 | 0.0096 | 1:07:31 |

TABLE VII
Current Analysis for Encrypted Data (MLP)

| Epoch | Accuracy (%) | | Loss | Time |
|---|---|---|---|---|
| | Testing | Training | Training | Training time (hr) |
| 1 | 94.35 | 97.81 | 0.0160 | 0:55:37 |
| 2 | 94.30 | 98.36 | 0.0114 | 0:55:39 |
| 3 | 94.20 | 98.65 | 0.0103 | 0:55:36 |
| 4 | 94.10 | 98.71 | 0.0097 | 0:55:41 |
| 5 | 94.10 | 98.8 | 0.0093 | 0:55:51 |

TABLE VIII
CURRENT ANALYSIS FOR ENCRYPTED DATA (RNN)

| | Accuracy (%) | | Loss | Time |
|---|---|---|---|---|
| Epoch | Testing | Training | Training | Training time (hr) |
| 1 | 86.40 | 94.22 | 0.0193 | 0:57:31 |
| 2 | 86.35 | 96.47 | 0.0120 | 0:56:39 |
| 3 | 86.25 | 96.12 | 0.0108 | 0:57:12 |
| 4 | 86.05 | 95.73 | 0.0101 | 0:57:35 |
| 5 | 85.80 | 95.38 | 0.0096 | 0:58:08 |

TABLE IX
CURRENT ANALYSIS FOR UNENCRYPTED DATA (MLP)

| | Accuracy (%) | | Loss | Time |
|---|---|---|---|---|
| Epoch | Testing | Training | Training | Training time (hr) |
| 1 | 88.00 | 97.81 | 0.0193 | 0:00:55 |
| 2 | 88.00 | 98.60 | 0.0120 | 0:00:58 |
| 3 | 87.90 | 96.70 | 0.0108 | 0:00:55 |
| 4 | 87.90 | 98.83 | 0.0101 | 0:01:15 |
| 5 | 88.00 | 99.91 | 0.0096 | 0:00:56 |

## REFERENCES


[1] P. Paillier, "Public-Key Cryptosystems based on Composite Degree Residuosity Classes," LNCS, vol. 1592, pp. 223–238, 1999, Accessed: Mar. 09, 2024. [Online]. Available: https://link.springer.com/content/pdf/10.1007/3-540-48910-X_16.pdf.

[2] corob-msft, "IEEE Floating-Point Representation," learn.microsoft.com.https://learn.microsoft.com/en-us/cpp/build/ieee-floating-point-representation?view=msvc-170

[3] data61, "data61/python-paillier," GitHub, Oct. 23, 2023. https://github.com/data61/python-paillier

[4] A. Bhayani, "How python implements super long integers?", Jan. 10, 2010. https://www.codementor.io/@arpitbhayani/how-python-implements-super-long-integers-12icwon5vk

[5] A. Hamed, "Exploring NumPy: Features and Performance Vs Lists," Medium, Aug. 30, 2023. https://blog.stackademic.com/exploring-numpy-features-performance-vs-lists-7f0b43d2af5f (accessed May 14, 2024).

[6] H. Inoue and T. Nakatani, "Performance of multi-process and multi-thread processing on multi-core SMT processors," IEEE International Symposium on Workload Characterization (IISWC'10), Dec. 2010, doi: https://doi.org/10.1109/IISWC.2010.5650174.

[7] A. Allharbi, S. Alhaidari, and M. Zohdy, "Denial-of-Service, Probing, User to Root (U2R) and Remote to User (R2L) Attack Detection using Hidden Markov Models," Sep. 2018.

[8] "KDD Cup 1999 Data," kdd.ics.uci.edu, Oct. 28, 1999. https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[9] K. Munjal and R. Bhatia, "Analysing RSA and PAILLIER homomorphic Property for security in Cloud," Procedia Computer Science, vol. 215, pp. 240–246, 2022, doi: https://doi.org/10.1016/j.procs.2022.12.027.

[10] T. Wingarz, M. Gomez-Barrero, C. Busch, and M. Fischer, "Privacy-Preserving Convolutional Neural Networks Using Homomorphic Encryption," in *Proceedings of the Conference*, 2022. [Online]. Available: https://doi.org/10.1109/iwbf55382.2022.9794535/

[11] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy," in *Proceedings of the International Conference on Machine Learning (ICML)*, 2016, pp. 201-210.

[12] IBM, "Memory utilization," www.ibm.com, Jul. 20, 2022. https://www.ibm.com/docs/en/informix-servers/14.10?topic=performance-memory-utilization

[13] L. Sgaglione et al., "Privacy Preserving Intrusion Detection Via Homomorphic Encryption," 2019 IEEE 28th International Conference on Enabling Technologies, Jun. 2019, doi:10.1109/wetice.2019.00073.

[14] J.-W. Lee et al., "Privacy-Preserving machine learning with fully homomorphic encryption for deep neural network," IEEE Access, vol. 10, pp. 30039–30054, Jan. 2022, doi:10.1109/access.2022.3159694.

[15] L. Jiang, C. Xu, X. Wang, and C. Lin, "Statistical learning based fully homomorphic encryption on encrypted data," Soft Computing, vol. 21, no. 24, pp. 7473–7483, Aug. 2016, doi:10.1007/s00500-016-2296-6.

[16] N. Dowlin, R. Gilad-Bachrach, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Manual for using Homomorphic Encryption for bioinformatics," Proceedings of the IEEE, pp. 1-16;, Jan. 2017, doi: 10.1109/jproc.2016.2622218.

[17] S. Bharati and P. Podder, "Machine and deep learning for IoT security and privacy: applications, challenges, and future directions," Security and Communication Networks, vol. 2022, pp. 1–41, Aug. 2022, doi: 10.1155/2022/8951961.

[18] N. Kaaniche and M. Laurent, "Data security and privacy preservation in cloud storage environments based on cryptographic mechanisms," Computer Communications, vol. 111, pp. 120–141, Oct. 2017, doi: 10.1016/j.comcom.2017.07.006.

[19] A. Ali, B. A. S. Al-Rimy, F. S. Alsubaei, A. A. Almazroi, and A. A. Almazroi, "HealthLock: Blockchain-Based privacy preservation using homomorphic encryption in internet of things healthcare applications," Sensors, vol. 23, no. 15, p. 6762, Jul. 2023, doi:10.3390/s23156762.