IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 January 2023, accepted 14 March 2023, date of publication 29 March 2023, date of current version 3 April 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3263114

RESEARCH ARTICLE

PPNNBP: A Third Party Privacy-Preserving Neural Network With Back-Propagation Learning

NAWAL ALMUTAIRI[1], FRANS COENEN[2], AND KEITH DURES[2]
[1]Information Technology Department, College of Computer and Information Science, King Saud University, Riyadh 11451, Saudi Arabia
[2]Department of Computer Science, University of Liverpool, L69 3DR Liverpool, U.K.

Corresponding author: Nawal Almutairi (nawalmutairi@ksu.edu.sa)

This work was supported by the "Research Center of the Female Scientific and Medical Colleges," Deanship of Scientific Research, King Saud University.

ABSTRACT With the advances in machine learning techniques and the potency of cloud computing there is an increasing adoption of third party cloud services for outsourcing training and prediction of machine learning models. Although cloud-hosted machine learning services enable more efficient storage and computation of data, privacy concerns and data sovereignty issues remain a major challenge. Privacy-preserving machine learning provides a promising solution. In this paper, a privacy-preserving neural network generation and utilization framework is presented, the PPNNBP framework. PPNNBP allows model training and prediction to be securely delegated to a third party with minimal data owner participation once the input data have been encrypted without recourse to secret sharing or multiple party setting. This is achieved using a proposed fully homomorphic encryption scheme, the Modified Liu Scheme (MLS), that permits certain operations over cyphertexts and features order preservation. The PPNNBP framework using MLS addresses the challenge of computational complexity of model learning using existing schemes; a complexity caused by the increasing size of cyphertexts (cyphertext inflation) and the quantity of noise introduced into cyphertexts through the application of multiplication operations, as learning progresses. Both the PPNNBP framework and MLS are fully described and analysed. The reported evaluation demonstrates that the PPNNBP framework achieves accuracy that is comparable to that obtained using a "standard" framework, whilst at the same time operating in a secure manner with minimal data owner participation.

INDEX TERMS Homomorphic encryption, secure machine learning as a service, secure neural network.

I. INTRODUCTION

The growth in public cloud service providers has encouraged the emergence of competitive services whereby cloud providers sell their computing power. Recently, many efforts have been directed at Machine Learning as a Service (MLaaS), where cloud providers offer model training and online prediction services to clients. MLaaS is currently provided by major organizations including Microsoft, Google, and Amazon [1], [2], [3]. For example, Google cloud Machine Learning (ML) engine allows data owners to upload their data that is used to train model in Tensorflow environment. Pre-trained models can be offered online, for any user, to download and fine-tuning such as Inception, AlexNet, and VGG [4], [5]. However, both of the aforementioned outsourcing strategies require access to the raw data which is often come with security concerns. There are various attacks directed to risk data and ML models such as Model Inversion Attack (MIA) [6], [7], [8]. These possible attacks demonstrate that both ML models and training datasets can be the target of privacy attacks, leading to sensitive information leakage. This problem tends to limit the take-up of MLaaS, especially in fields where data disclosure is not only a commercial privacy problem, but also a legal concern [9], [10]. There is thus a requirement for techniques that provide rigorous privacy guarantees to data owners, whereas providing the utility to support MLaaS. This paper presents privacy-preserving Neural Network (NN) training and usage.

The associate editor coordinating the review of this manuscript and approving it for publication was Yu-Chi Chen.

Data encryption can substantially guarantee data privacy, but precludes any data manipulation. The introduction of Fully Homomorphic Encryption (FHE) schemes, that feature a number of mathematical operations that can be applied directly to cyphertexts (without decryption), has offered a potential means of achieving secure ML [11], [12], [13]. However, although FHE may address the privacy concerns associated with the wide-scale adoption of MLaaS, it features three principal disadvantages. The first, and most significant, is the number of operations supported by FHE schemes, typically limited to addition and multiplication, which in turn means that more sophisticated operations, such as numeric comparison, require data owner (or key holder) participation; for many applications the amount of participation is substantial [13], [14], [15]. The second disadvantage is that the size of the cyphertexts increases exponentially with the application of each multiplication operation, as a consequence of which large amounts of "noise" are also introduced into the cyphertexts. This is typically resolved, in many FHE schemes, by some form of noise management techniques such as bootstrapping, modulus switching, scale invariant and flattering [11], [16]. These techniques, however, tend to be complex [17], [18]. Recently, noise-free FHE schemes have been proposed whereby noise management can be avoided by allowing an arbitrarily large quantity of noise [19], [20]. Most of these schemes rely on multivariate quadratic and univariate high degree polynomial equation systems [21], [22]. The multivariate and high degree polynomial equations, in noise-free FHE schemes, have introduced a cypher inflation problem whereby the number of subcyphers used to encode a plaintext increases exponentially with each multiplication. This has raised problems concerning computational cost and memory resource requirements [20]. The third disadvantage is that the defined message space for most practical FHE schemes is restricted to either the binary [23], [24], [25] or positive integer space [26], [27], [28]. This means that the direct encryption of real number values is not supported; introduced solutions adversely affect the accuracy of any learnt model [29], [30], [31], [32]. As a consequence of these disadvantages, the potential of FHE in terms of secure MLaaS, has not been fully realised.

The PPNNBP framework introduced in this paper is directed at the secure training, and usage, of NN models using only encrypted data, in a manner that avoids the disadvantages associated with existing FHE schemes as described above. The framework achieves this by using a novel FHE scheme, the Modified Liu Scheme (MLS), founded on the Liu homomorphic encryption Scheme (LS) [20]. The MLS maintains the noise-free feature of LS, whereas at the same time providing a mechanism to address the cypher inflation problem that occurs after each multiplication operation as learning progresses; thus addressing the computational/resources cost problems. In addition, the MLS includes cyphertext order preservation to allow secure data comparison, hence avoiding data owner participation or recourse to complex data comparison protocols [14], [33], [34]; the first FHE scheme to do so. As in LS, MLS preserves the feature of directly encrypting real numbers. The features of MLS can be used to implement a variety of MLaaS services.

The main contribution of this paper is the PPNNBP framework that can be used to securely generate NN models using Back-Propagation (BP) learning. PPNNBP relies on a single cloud server compared to other solutions which require two [37], [38] or three [33] servers. The paper also proposes a secure linear approximation of the nonlinear NN activation function. The underlying reasons for selecting NN with BP learning are as follows: (i) it is a more powerful learning algorithm than (say) the linear regression and logistic regression ML algorithms; and (ii) the BP learning method is a central feature of more sophisticated Deep NN (DNN) and hence the PPNNBP framework presented here "paves the way" to privacy-preserving DNN. The PPNNBP framework therefore allows for the secure training of NNs over encrypted data; the training data, weights, biases and activation parameters are all encrypted and only the learning rate and momentum are given in plaintext form. Similarly, the usage of the trained NN is conducted in an encrypted manner (both input data and predicted output). The PPNNBP framework features only minimal data owner participation during training and usage. Unlike alternative frameworks [14], [17], [18], [35], PPNNBP achieves an accuracy comparable with that of identical networks trained without encryption.

The rest of this paper is organized as follows. In Section II, related works are reviewed. Section III outlines an attack model. The peculiarities of the introduced MLS are presented in Section IV. Section V proposes methods to approximate sigmoid function as a low degree polynomial. Next, the extension of NN with BP learning to preserve privacy using the MLS is discussed. Section VII is devoted to experimental results and evaluation. Finally, some concluding remarks and future work are given in Section VIII.

II. RELATED WORK

MLaaS over encrypted data has been investigated with the respect to many ML algorithms. In the context of NN there are two MLaaS variations; Training as a Service (TaaS) and Prediction as a Service (PaaS). In TaaS, a cloud provider sells services that allow data owners to upload their encrypted data and receive a trained NN model [18], [33], [36], [37], [38], [41]. For preserving data privacy, the trained model weights, biases and intermediate calculations as activation functions are encrypted. In PaaS, a pre-trained NN model hosted by a cloud provider is monetised for labelling clients instances in such a way that the data privacy preservation is maintained for data instances, predicted labellings and network model weights and biases [14], [17], [33], [35], [37], [38]. The disadvantages of using FHE for both TaaS and PaaS were highlighted in the introduction to this paper. In the reminder of this section a number of solutions to address these disadvantages are reviewed. Each offers a potential solution but at a cost; costs addressed by the solution presented in this paper.

From the literature it can be observed that the main focus of recent work with the domain of secure MLaaS has been directed at secure PaaS [14], [17], [33], [35], [37], [38]. In many cases the learning method was executed over plaintext training data and the (unencrypted) trained model was then used to provide PaaS [14], [35]. The challenge was how to "feed-forward" a network with private client instances (queries), and calculate the nonlinear activation functions, without revealing the network's weights and biases to the client. The straight forward solution relied on Secure Multi-Parity Computation (SMPC) [33], [37], [38] and FHE scheme [14], [37]. Using this solution, the clients encrypt their private data instances using an appropriate FHE scheme. The PaaS provider uses the encrypted instances to feed a NN. First the inner products between the encrypted data instance and unencrypted weights of the first layer are computed and then sends the encrypted inner products to the client. The client decrypts the products, applies the nonlinear activation function and encrypts the result before sending it back to the PaaS provider who repeats the process for the remaining layers. This approach addresses the complexity of evaluating the activation function but introduces three major limitations: (i) a significant and undesirable computational overhead on behalf of the client, (ii) high latency and high bandwidth usage, and (iii) undesirable disclosure of the network weights and biases to the client. The last introduces a security concern whereby a MIA [8] can be used to reveal confidential aspects of the data originally used to train the model. The research presented in [30] therefore suggested a mechanism whereby the weights and biases could be obscured using a Oblivious Transformation (OT) technique that added noises before delegating the evaluation of activation functions to the client. However, the solution had a similar communication/computation overhead as in the case of [14]. Secure PaaS mechanisms that do not require recourse to clients, as in the case of [14] and [30], typically work by using some form of alternative activation functions that can operate in encrypted form. A popular choice here is a quadratic activation function, $f(x) = x^2$, that can be evaluated using FHE properties as presented in [18], [35], and [39]. However, the quadratic function results in accuracy loss [40]. Whatever the case, the replacement of the activation function does not address the substantial involvement of clients or data owners, in PaaS or TaaS, as FHE schemes typically required the execution of noise management techniques or required the re-encrypting of cyphertexts when noise exceeds a predetermined level [17], [18], [35]; this is clearly undesirable. To reduce the amount of time required to re-encrypt cyphertexts (the principal communication overhead), the level of the FHE scheme used can be increased [26]. However, this will increase the size of the cyphertexts, and thus adversely reduce the scheme multiplication efficiency [41]; this is clearly also undesirable.

Secure TaaS using encrypted data has been considered in [18], [36], [37], [38], [41], and [33]. Here the model was trained using encrypted data, and encrypted network weights and biases are generated. In [33], [37], and [38], the learning was facilitated by: (i) splitting the training data across non-colluding cloud servers who will jointly run SMPC protocols; (ii) using secure inner product calculation with respect to the multiplication of network weights and training data features; and (iii) Yao's comparison protocols to evaluate comparisons. Two non-colluding cloud servers are required with respect to the work presented in [37] and [38], and three servers with respect to the work presented in [33]. However, this form of secure TaaS using encrypted data has three major limitations: (i) the computational complexity of SMPC protocols, (ii) the requirement for at least two non-colluding cloud servers, and (iii) the high operating cost of utilizing multiple cloud servers. The work in [41] presents several approaches for TaaS within the practical limitations of existing Homomorphic Encryption (HE) schemes (without resorting to SMPC). The idea was to replace computations with equivalent HE properties and approximate well-established activation function. The replacement of activation functions with quadratic function, as in the case of PaaS, is not appropriate for TaaS as it features "unbounded derivation" that could result in unusual behavior of the trained model [40]. The alternative is to approximate established activation functions using a linear polynomial; Chebyshev polynomials are used in [41] and [18] whereas Taylor polynomials are used in [17] and [43]. Experiments have demonstrated that higher degree polynomials give a better approximation and thus a better replacement for established activation functions. The Rectified Linear Unit (ReLU) activation function was approximated in [17] by using degree six polynomials and the sigmoid activation function using degree three polynomials. However, the disadvantages of using these approximation were: (i) when using higher degree polynomials the polynomial coefficients become very small (for example $\times 10^{-31}$), these are usually truncated to a small number of digits because of technical limitations, which in turn affects accuracy, (ii) the amount of noise added within the cyphertexts, after each homomorphic multiplication, makes data owner (or client) participation mandatory to re-encrypt cyphertexts when the noise exceeds a pre-defined level or requires complex noise management techniques, and (iii) the approximation requires the pre-analysis of the input data to decide the appropriate intervals, their corresponding approximation and their polynomial coefficients.

There are a limited number of frameworks that provide both secure TaaS and secure PaaS, some of which are listed in Table 1. The frameworks use a variety of cryptographic methods to maintain data privacy, including the SMPC [33], [37], [38], OT [37], [38], and the Leveled HE (LHE) scheme, which supports additions and limited multiplications [37]. The following four limitations can be identified from these frameworks: (i) the requirement to represent data using fixed points instead of decimal values, requiring the use of fixed arithmetic for operations such as replacing activation

TABLE 1. Comparison of the privacy-preserving frameworks that offer both TaaS and PaaS. DO: Data Owner and SGD: Stochastic Gradient Descent.

Framework	Crypto. Methods	Activation Function	Decimal / Fixed Arithmetic	# of Cloud Servers	SS	Disclosure of Inferencing Results	Disclosure of NN Topology	DO Involv.	Disclosure of Intermediate Results
SecureNN [33]	SMPC	ReLU	Fixed 13-bit	3 Servers	✓	Client only	Servers	×	✓
SecureML [37]	LHE, OT & SMPC	ReLU, Softmax & quadratic	Decimal point	2 Servers	✓	Client only	Servers	✓	✓
QUOTIENT [38]	OT & SMPC	ReLU	Fixed 8-bit SGD 2-bit Weight	2 Servers	✓	Client only	Servers	✓	✓
PPNNBP	FHE	Sigmoid & Linear	Decimal point	1 Server	×	Client only	Server	✓	×

functions, (ii) the number of servers involved and thus the associated security assumption, (iii) the requirement that data owners participate, and (iv) the disclosure of intermediate results. All secure frameworks, presented in [33], [37], and [38], require multiple servers that rely on the Secret Sharing (SS) to facilitate the secure NN learning. Regardless of the specific role assigned to the servers, the trust model assumes that the two servers are untrusted but do not collude; which is often considered a security risk. The SS is used to reduce or in some cases avoid data owner participation. This is achieved by allowing multiple parties to collaboratively perform any require operations on behalf of data owner which in turn disclosure the intermediate results. In this paper, only one cloud server is used to train the model without recourse to SS or relying on unrealistic security assumption. For this to be achieved, it requires more efficient encoding scheme that allows for faster homomorphic computation and serve to address FHE limitations. The MLS, described in detail later in this paper, avoids the above disadvantages, and allows: (i) unlimited homomorphic multiplications without the problem of cypher inflation and without recourse to some form of noise management; (ii) PaaS over encrypted data without involving any data owner participation, whilst the prediction process is progressing; and (iii) a dramatic reduction of the amount of data owner participation for TaaS whereas maintaining model accuracy. As in the case of [33], [37], and [38] frameworks, in PPNNBP the network topology is disclosed to the server and the prediction result is only revealed to the client.

III. ATTACK MODEL

In terms of attack model categorisation, the cloud provider is assumed to be a *passive adversary* who follows the *semi-honest attack model*. This means we assume that the NN training and prediction algorithms will be honestly executed (this is, after all, in the commercial interest of the provider), but at the same time attempts may be made to learn additional information by analysing the encrypted data received or encrypted intermediate data produced during the execution. The potential attacks that can be directed at the PPNNBP system are: (i) Cyphertext Only Attacks (COAs) and (ii) MIA [8]. A COA is where an attacker only has access to encrypted data. In the case of the PPNNBP system this might be: the training data, client data instances, intermediate calculations, predicted labels and/or the trained model weights and biases. In the case of MLS, a COA might be used to exploit the ordering feature of MLS cyphers and extract statistical measures describing the frequency of distribution patterns, that might then be used to identify the nature of the plaintext values. However, this will only succeed if the attacker has previous knowledge concerning the original data. A MIA is where an attacker has access to the NN model and is able to utilise PaaS with the intention of acquiring information concerning the model's behavior beyond simply the prediction results. In this attack, and as noted in [8], the attacker can exploit the predictions to reveal confidential aspects of the data originally used to train the NN model.

IV. MODIFIED LIU SCHEME (MLS)

The MLS, utilised by the PPNNBP, is a new scheme that modifies the original LS presented in [20]. The modifications incorporated into MLS had two primary objectives: (i) addressing the cyphertext inflation problem that occurs whenever homomorphic multiplication is applied using the concept of *trapdoors*; and (ii) providing an ordering feature in the generated cyphertexts so as to allow encrypted data comparison using what is referred to as the *ω-concept*, the idea of including a "gap" between subcyphertexts so that different cyphertexts can be generated for the same plaintext value whereas data ordering is preserved (but not data equality). MLS still retains the same characteristics and homomorphic properties of the original LS, and thus is noise-free, and supports both addition ($\oplus$) and multiplication ($\otimes$) over cyphertexts, and the multiplication of cyphertexts with plaintexts values ($\circledast$). Subtraction operation ($\ominus$) can be implemented using multiplication with plaintext value ($\circledast$ -1) and additive property ($\oplus$). The message space and cyphertext space are as defined for the original scheme; $\mathbb{R}$ and $\mathbb{R}^m$ respectively. This means direct encryption of real values are supported. The following subsections, Subsections IV-A to IV-E, present the

MLS scheme algorithms and processes for; key generation, trapdoor calculation, data encryption/decryption and cypher inflation prevention. Note that in the remainder of this paper the cyphertext equivalent of a plaintext value x is given by $E(x)$ which, for simplicity, is written as x'.

A. KEY GENERATION

The same Secret Key (SK) configuration as used in the original LS is used in MLS, SK(m)= $[(k_1, s_1, t_1), \ldots (k_m,s_m, t_m)]$. The difference is that the values for the k and s components are split into two parts, a "secret key" and "shared key". The secret part of the key is kept locally by the data owner, whereas the shared part is used to calculate trapdoors that allows the desired subcypher "dimensionality reduction" (addressing the cypher inflation problem as discussed further in Subsections IV-B and IV-E below). The first step required to generate the secret key is to randomly select values for SK(m) in such a way that the following conditions are satisfied:

1) As in the case of LS, the number of subcyphertexts generated by the MLS is m where $m \geqslant 3$.
2) $k_m + s_m + t_m \neq 0$.
3) k_i and s_i are positive integers ($1 \leqslant i \leqslant m$) and the GCDs (Greatest Common Divisors) for k_i and s_i are > 1 and not equal to s_i or k_i.
4) There exists only one element q ($1 \leqslant q < m$) such that $t_q \neq 0$. This condition was introduced in [42] for facilitating secure data comparison in a secure k-Means data clustering context. In MLS $t_q = (s_q+k_q) \times \omega$, where ω is the numeric gap between cyphertexts included so that ordering is preserved. The ω value adopted was 10^p, selected to create a large gap that permits increasing the number of cyphertexts that can be generated for the same plaintext value (a nondeterministic feature).

The list of random numbers $R = [r_1, \ldots, r_{m-1}]$, used for encryption purposes together with the secret key, are all random nonzero positive numbers between 1 and ω; selected in such a way that r_q, corresponding to element q in the secret key, is greater than all the remaining random values.

B. TRAPDOORS CALCULATION

Trapdoors, as noted above, are used for "dimensionality reduction". There is one set of trapdoors, Trap = [trap$_1, \ldots,$ trap$_m$], associated with a single secret key, and there is a one-to-one correspondence between the two. The last element of the list, trap$_m$, as will be demonstrated later, is of particular significance and is designated as the *kst* value and is calculated separately; thus for practical purposes Trap = [trap$_1, \ldots,$ trap$_{m-1}$]. The process for producing Trap is given by Algorithm 1. The algorithm commences by calculating the GCD of the subkeys s and k to be retained locally by the data owner (lines 2 and 3). The set Trap and the shared sets, SharedS and SharedK, are then defined in lines 4 and 5 as sets of $m - 1$ elements. The set Trap holds the trapdoor values, whilst SharedS and SharedK hold the shared part of the secret key used to calculate the trapdoor values held in Trap. The algorithm then loops from $i = 1$ to $i = m - 1$ (lines 6 to 9) to calculate the shared part of the key, SharedS and SharedK, that are then used to calculate the trapdoors as per the equations given in lines 7 to 9. The kst value is then calculated as per equation in line 10. The algorithm exits with Trap and the kst value (line 11).

Algorithm 1 MLS Trapdoor Calculation

```
1:  procedure TrapdoorsCalculation(SK($m$))
2:      secretS= GCD($s_1, \ldots, s_{m-1}$)
3:      secretK= GCD($k_1, \ldots, k_{m-1}$)
4:      Declare Trap as a set of $m-1$ elements
5:      Declare SharedS and SharedK as set of $m-1$
    elements
6:      for $i = 1$ to $i = m-1$ do
7:          sharedS$_i$ = $\frac{s_i}{secretS}$          ▷ sharedS$_i$ ∈ SharedS
8:          sharedK$_i$ = $\frac{k_i}{secretK}$          ▷ sharedK$_i$ ∈ SharedK
9:          trap$_i$ = $\frac{sharedS_i}{sharedK_i}$          ▷ trap$_i$ ∈ Trap
10:     kst = $k_m + s_m + t_m$
11:     Exit with Trap and kst
```

C. ENCRYPTION

The MLS encryption function uses SK(m) to convert a value x to m subcyphertexts $E(x) = \{e_1, \ldots, e_m\}$ following steps very similar to those adopted in LS as shown in Algorithm 2. The variable l, in line 8, is the cyphertext level counter, the number of times that dimensionality reduction has been applied to the cyphertext. There is no limit for the number of levels supported by the MLS, however, the value of l is required for decryption purposes (see Subsection IV-D below). The MLS encryption function associated with the conditions defined by the key generation conditions presented in Subsection IV-A preserve the order of the plaintext value in the qth subcypher (e_q). The proof of correctness is given in Appendix A.

Algorithm 2 MLS Encryption

```
1:  procedure Encrypt($x$,SK($m$))
2:      Uniformly generate $m-1$ arbitrarily random
    numbers $R = \{r_1, \ldots, r_{m-1}\}$
3:      Declare $E$ as a list of $m$ elements $E = \{e_1, \ldots, e_m\}$
4:      $e_1 = \frac{(k_1 \times t_1 \times x + s_1 + k_1 \times (r_1 - r_{m-1}))}{s_1}$
5:      for $i = 2$ to $i = m-1$ do
6:          $e_i = \frac{(k_i \times t_i \times x + s_i + k_i \times (r_i - r_{i-1}))}{s_i}$
7:      $e_m = k_m + s_m + t_m$
8:      $E.l = 0$
9:      Exit with $E$
```

D. DECRYPTION

The decryption function decodes a cyphertext E to its plaintext equivalent x, following a process very similar to the original LS, as shown by Algorithm 3. The algorithm starts by calculating the value for t (line 2). The algorithm then calculates the new subcyphertext value for each subcypher e_i in E once the dimensionality of the cyphertext has been reduced (lines 3 to 5). This step is required to produce a correct decryption. The level counter and secret part calculated in Algorithm 1 are used to return a cyphertext value to its value before performing dimensionality reductions. The algorithm then calculates the s value, line 6, which is then used in line 7, with SK(m) and t, to calculate the decoded value x. The algorithm will exit in line 8 with the decoded (plaintext) value.

Algorithm 3 MLS Decryption

```
1: procedure Decrypt($E$, SK($m$), $secretS$, $secretK$)
2:     $t = \sum_{i=1}^{m-1} t_i$
3:     if $E.l \neq 0$ then
4:         for $i = 1$ to $i = m$ do
5:             $e_i = \frac{(e_i \times (secretS^{E.l}/secretK^{E.l}))}{t^{E.l}}$
6:     $s = \frac{e_m}{(k_m + s_m + t_m)}$
7:     $x = \frac{(\sum_{i=1}^{m-1}((e_i \times s_i) - (s \times s_i))/k_i)}{t}$
8:     Exit with $x$
```

E. SUBCYPHER DIMENSIONALITY REDUCTION

In MLS, as in LS, cyphertext multiplication is achieved by determining the outer product of the two cyphertexts. Given two plaintext values x_1 and x_2, these are encrypted using MLS and SK(m), to give $E_1 = \{e_{1_1}, \ldots e_{1_m}\}$ and $E_2 = \{e_{2_1}, \ldots, e_{2_m}\}$ respectively. The cyphertext multiplication $E_1 \otimes E_2$ is implemented as: $\{e_{1_1}, \ldots, e_{1_m}\} \otimes \{e_{2_1}, \ldots, e_{2_m}\} = \{e_{1_1} \times e_{2_1}, \ldots, e_{1_1} \times e_{2_m}, \ldots, e_{1_m} \times e_{2_1}, \ldots, e_{1_m} \times e_{2_m}\}$. Therefore, for one multiplication the cyphertext size (dimensionality) is increased from m to m^2 and continues to exponentially increase with each multiplication operation. This cyphertext inflation, as noted earlier, causes a computational overhead and also leads to a scalability problem. Using the MLS the size of the generated cyphertext, after a multiplication operation, is "reduced" back to m using trapdoor information that allows re-encryption of the cyphertext (without prior decryption); this is the "dimensionality reduction" referred to earlier.

Algorithm 4 presents the pseudo code for the dimensionality reduction process. The algorithm takes as inputs: (i) a sequence of subcyphertexts $E = \{e_1, \ldots, e_{m^2}\}$, (ii) a set of trapdoors Trap, and (iii) the kst value. The algorithm commences (line 2) by declaring a reduced cyphertext list RE of length m. Next, an index j for the cyphertext set E and an index z for the reduced cyphertext RE are declared and initialised (line 3). The algorithm then loops through m^2 subcyphertexts in E (lines 4 to 12). Each iteration commences (line 5) with the creation of a temporary cyphertext, Temp$'$, made up of m subcyphertexts in E started by jth index. The mth subcyphertext in Temp$'$ and the trapdoor value kst are used to calculate the value for the parameter s' (line 6). The algorithm then (line 7) defines the variable subCypher in which to hold the current subcyphertext value once calculated. Next, the algorithm loops through Temp$'$ (lines 8 to 10) and determines the new subcyphertext value and, on completion, appends it to the list RE which holds the cyphertexts as calculated so far. The new subcyphertext values are calculated using the FHE properties of the MLS scheme; addition $\oplus$, subtraction $\ominus$ and multiplication $\circledast$. The values of indexes, z and j, are then updated in line 12. Next, line 13, the cypher level counter is incremented by one, $E.l + 1$. At the end of the process the newly calculated cyphertext, of length m, is returned (line 14). In the remainder of this paper the multiplication of two cyphertexts, followed by dimensionality reduction, is indicated using the operator $\bigotimes$; whereas multiplying a cyphertext with a plaintext value is indicated using the operator $\circledast$. The correctness of dimensionality reduction algorithm (Algorithm 4) is given in Appendix A.

Algorithm 4 Dimensionality Reduction Process

```
1: procedure DimReduction($E$, Trap,kst)
2:     Declare RE as a list of $m$ elements
3:     $j = 1, z = 1$
4:     while $j < m^2$ do
5:         Temp$'$ = Copy subcyphertext in $E$ started by $j$th
           index of length $m$
6:         $s' = \text{temp}'_m \circledast \frac{1}{kst}$                ▷ $\text{temp}'_m \in \text{Temp}'$
7:         subCypher= 0
8:         for $i = 1$ to $i = m - 1$ do
9:             $t' = \text{temp}'_i \ominus s' \circledast \text{trap}_i$          ▷ $\text{trap}_i \in \text{Trap}$
10:            subCypher=subCypher $\oplus t'$
11:        $\text{re}_z$ = subCypher                          ▷ $\text{re}_z \in \text{RE}$
12:        $z = z + 1, j = j + m$
13:    RE.$l = E.l + 1$
14:    Exit with RE
```

V. POLYNOMIAL APPROXIMATION OF ACTIVATION FUNCTION

The sigmoid activation function, given in (1), is a nonlinear function that can not be directly computed using the mathematical properties of FHE schemes [17], [18], [35], [36], [37]. The operation of the sigmoid activation function can be approximated, up to a certain accuracy, using a polynomial approximation method that uses: (i) Taylor series expansions [17], [43], (ii) Chebyshev polynomials [18], [41], and (iii) Maclaurin series expansions [44]. In practice, this approximation needs to be done using a high degree polynomial for accurate results to be obtained. This in turn increases the number of HE multiplications, the amount of noise, and the size of the cyphertexts. In this paper, two sigmoid approximations are utilised, the proposed TaylorLinear (φ) and FriendlyFunction (ϕ) as presented in [37]. With respect to the PPNNBP, TaylorLinear approximation was used for training the PPNNBP (TaaS), with limited data owner participation,

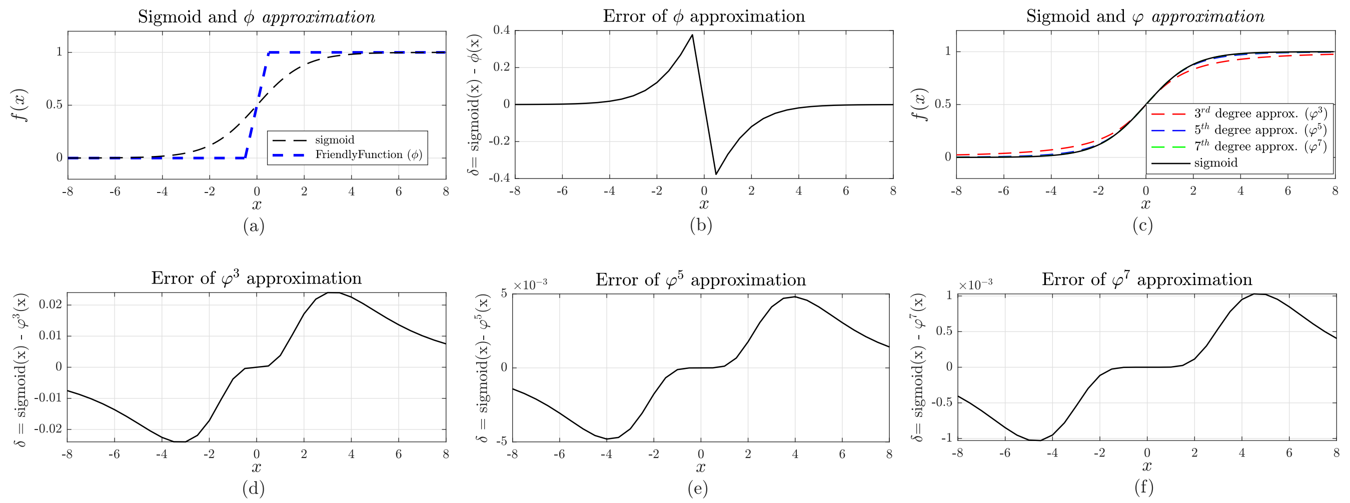

FIGURE 1. **Comparison of the FriendlyFunction (ϕ) and TaylorLinear (φ) approximations and their error of approximation compared to the sigmoid function.**

and the FriendlyFunction to provide query classification once the NN had been trained (PaaS).

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

The TaylorLinear approximation offers the advantage that it maintains accepted accuracy levels and can be implemented using MLS with very limited data owner participation. The Taylor series expansion is used to linearly approximate the e^{-x} term, which is a part of the sigmoid activation function as in (1). The e^{-x} term is approximated as per (2) where d is the degree of polynomial selected by the data owner according to the required level of accuracy. Using these parameters TaylorLinear approximates sigmoid, $\varphi(x')$ where x' is MLS cyphertext, as follows:

1) Using FHE properties of MLS, the cloud service provider calculates the Taylor approximation $1 + e^{|x'|}$ using the Taylor polynomial given in (3) where d represents the degree of the function as selected by the data owner according to the required accuracy set against execution time (there is an inverse trade-off).
2) The data owner performs the "inversion" of value $1 + e^{|x'|}$ to arrive at the approximated value $\frac{1}{1+e^{|x'|}}$.
3) If $x' \geqslant E(0)$ the cloud service provider will calculate the activation function as $1 - \frac{1}{1+e^{|x'|}}$. Otherwise the activation function is as approximated in step 2.

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots + \frac{(-1)^d x^d}{d!} \tag{2}$$

$$1 \oplus (1 \oplus x' \oplus \frac{1}{2!} \circledast (x' \bigotimes x') \oplus \frac{1}{3!} \circledast (x' \bigotimes x' \bigotimes x') \oplus \cdots \oplus \frac{1}{d!} \circledast (x' \bigotimes \ldots \bigotimes x')) \tag{3}$$

The absolute value of x', $|x'|$, used in step 1 is calculated by multiplying x' with -1, using $\circledast$, when cypher x' is less than the MLS cyphertext of zero. The comparison of x' with zero (in step 3) is conducted, using the MLS properties, by comparing the qth subcyphertext of x' with qth subcyphertext of zero encrypted. Step 3 also relies on a mathematical rule associated with the sigmoid function that allows the calculation of sigmoid(x) and sigmoid($-x$) as per equation $sigmoid(-x) = 1-sigmoid(x)$. The TaylorLinear requires some data owner participation; but this participation is minimal compared with alternative approaches such as those given in [17] and [41] as described earlier in Section II.

A secure activation function is also required in the context of the provision of PaaS. TaylorLinear can again be used for this purpose. Alternatively FriendlyFunction (ϕ) linear approximation, as considered in [37], may be used. This offers the advantage, using the MLS, that it can operate over encrypted data without any data owner participation. The FriendlyFunction is a piecewise-linear approximation that returns 0 when $x < -0.5$, 1 when $x > 0.5$ and $x + 0.5$ when $-0.5 \leqslant x \leqslant 0.5$. However, it is not as accurate as TaylorLinear. Fig. 1, (a) and (c), shows a comparison of sigmoid activation and its approximation using FriendlyFunction and TaylorLinear with a range of values for d, $d = \{3; 5; 7\}$ and different values for the input x. The symbol φ^d is used to refer to the TaylorLinear approximation function where d is the degree of polynomial. The figure also shows the error (δ) associated with the approximations; calculated as the difference between the sigmoid function and the approximated functions. The experiments show that the error of TaylorLinear when $d = 3$ is $\delta \in [-0.024; 0.024]$, greater than when $d = 5$ and $d = 7$, $\delta \in [-0.0048; 0.0048]$ and $\delta \in [-0.0010; 0.0010]$ respectively. The FriendlyFunction provides the worst case; $\delta \in [-0.40; 0.40]$. Therefore, the TaylorLinear approximation provides a better fit with the sigmoid function than FriendlyFunction approximation and thus may provide better TaaS and PaaS (although TaylorLinear requires some user participation). Moreover, TaylorLinear approximation presented in this paper can be used to obtain efficient protocols to train ML models that require calculation of sigmoid activation function.

VI. PRIVACY-PRESERVING BACK-PROPAGATION

The PPNNBP approach comprises a multilayer feed-forward network with BP learning which is both trained and used over encrypted data. The privacy-preserving TaaS (PPTaaS) is given in Algorithm 5 that calls Algorithms 6 for feed-forward and Algorithms 7 for BP. The notation used is presented in Table 2. The inputs for PPTaas are: (i) a set of encrypted training records $D' = \{r'_1, \ldots, r'_n\}$, each record r'_i features a set of a attributes $\{r'_{i,1}, \ldots, r'_{i,a}\}$; (ii) a maximum number of epochs, *maxEpoch*; (iii) a learning rate η; (iv) a momentum μ; (v) a set of encrypted target class (data) labels $T' = \{t'_1, \ldots, t'_n\}$ corresponding to the records in D', each featuring c binary attributes $t'_i = \{t'_{i,1}, \ldots, t'_{i,c}\}$, where c is the number of classes, and only one attribute has the cyphertext of 1 indicating the class for the record $r'_i \in D'$; (vi) an input-hidden-output network topology defined by $L = \{l_1, \ldots, l_b\}$ where l_1 is always equals to number of attributes in dataset and l_b is equals to number of class labels, (vii) a threshold error ϵ', and (viii) a d value for φ approximation; this will not be required if using ϕ approximation. The training data D', set of class labels T' and the error threshold ϵ' are all encrypted using the MLS. The outputs consist of weights W' and biases Θ' encoded in MLS for the network described by L.

Algorithm 5 Privacy-Preserving Training as a Service (PPTaaS)

```
1: procedure PPTaaS($D'$, $maxEpoch$, $\eta$, $\mu$, $T'$, $L$, $\epsilon'$, $d$)
2:     Initialize $W'$ and $\Theta'$ randomly and encrypt values using MLS
3:     Initialize $\Delta W'$=Encrypt(0) and $\Delta\Theta'$=Encrypt(0)
4:     $\delta'$=Encrypt(0)
5:     for $epoch = 1$ to $epoch = maxEpoch$ do
6:         for $s = 1$ to $s = n$ do
7:             $Y'$=PPFF($r'_s$, $L$, $d$, $W'$, $\Theta'$)    ▷ Algorithm 6
8:             $\delta'$,$W'$,$\theta'$=PPBP($Y'$,$t'_s$,$L$,$\delta'$,$W'$,$\Theta'$)    ▷ Algorithm 7
9:         $\delta' = \frac{1}{2 \times n} \circledast \delta'$
10:        if $\delta' < \epsilon'$ then
11:            Exit with $W'$ and $\Theta'$
12:    Exit with $W'$ and $\Theta'$
```

Algorithm 6 Privacy-Preserving Feed-Forward (PPFF)

```
1: procedure PPFF($r'$, $L$, $d$, $W'$, $\Theta'$)
2:     for $i = 1$ to $i = l_1$ do
3:         ${y'}_i^1 = r'_i$
4:     for $j = 2$ to $j = |L|$ do
5:         for $i = 1$ to $i = l_j$ do
6:             $v' = ({w'}_{1\,i}^{j-1} \bigotimes {y'}_1^{j-1}) \oplus \cdots \oplus ({w'}_{l_{j-1}\,i}^{j-1} \bigotimes {y'}_{l_{j-1}}^{j-1})$
7:             ${y'}_i^j = \varphi^d(v' \oplus {\theta'}_i^j)$
8:     Exit with $Y'$
```

Algorithm 7 Privacy-Preserving BP (PPBP)

```
1: procedure PPBP($Y'$,$t'_s$,$L$,$\delta'$,$W'$,$\Theta'$)
2:     one'=Encrypt(1)
3:     for $i = 1$ to $i = l_b$ do    ▷ $l_b$: # of neurons in output layer
4:         $e' = ({y'}_i^b \ominus t'_{s,i})$
5:         ${\delta'}_i^b = e' \bigotimes ({y'}_i^b \bigotimes (\text{one}' \ominus {y'}_i^b))$
6:     for $j = |L| - 1$ to $j = 2$ do
7:         for $i = 1$ to $i = l_j$ do    ▷ $l_j$: # of neurons in layer $j$
8:             ${\delta'}_i^j = {y'}_i^j \bigotimes (\text{one}' \ominus {y'}_i^j) \bigotimes [({w'}_{i\,1}^j \bigotimes {\delta'}_1^{j+1}) \oplus \cdots \oplus ({w'}_{i\,l_{j+1}}^j \bigotimes {\delta'}_{l_{j+1}}^{j+1})]$
9:     for $j = |L| - 1$ to $j = 2$ do
10:        for $i = 1$ to $i = l_j$ do
11:            $\Delta{\theta'}_i^j = (\eta \circledast {\delta'}_i^j) \oplus (\mu \circledast \Delta{\theta'}_i^j)$
12:            ${\theta'}_i^j = {\theta'}_i^j \oplus \Delta{\theta'}_i^j$
13:            for $k = 1$ to $k = l_{j-1}$ do
14:                $\Delta{w'}_{i\,k}^j = (\eta \circledast {\delta'}_i^j \bigotimes {y'}_k^{j-1}) \oplus (\mu \circledast \Delta{w'}_{i\,k}^j)$
15:                ${w'}_{i\,k}^j = {w'}_{i\,k}^j \oplus \Delta{w'}_{i\,k}^j$
16:    $Error' = [({y'}_1^b \ominus t'_{s,1}) \bigotimes ({y'}_1^b \ominus t'_{s,1})] \oplus \cdots \oplus [({y'}_c^b \ominus t'_{s,c}) \bigotimes ({y'}_c^b \ominus t'_{s,c})]$
17:    $\delta' = \delta' \oplus Error'$
18:    Exit with $\delta'$,$W'$,$\theta'$
```

As in the case of standard training using BP learning [45], the privacy-preserving TaaS with BP learning is composed of two stages: (i) *Privacy-Preserving Feed-Forward (PPFF)* and (ii) *Privacy-Preserving BP (PPBP)*. The algorithm commences, line 2, by defining the encrypted sets W' and Θ' and initialising them with random values; and then, line 3, defines the encrypted gradient of the loss with respect to the weights ($\Delta W'$) and gradient of the loss with respect to the biases ($\Delta\Theta'$) and initialising them with the value 0 encrypted using MLS. In line 4 the overall error value so far (the overall loss function), δ', is then defined and initialised with the MLS encrypted equivalent of zero. The training is then commenced (lines 5 to 11), the algorithm iterates until the specified maximum number of epochs is reached (*maxEpoch*), or the error δ' value becomes less than the error threshold ϵ'. On each iteration each sample in the encrypted dataset, $r'_s \in D'$, is processed in turn (lines 6 to 8) through calling PPFF and then PPBP. PPFF algorithm returns a set of encrypted outputs, Y', for all neurons in NN as specified in topology L. The PPBP takes the resulting output from PPFF (Y'), encrypted target label t'_s, encrypted weights and bias (W' and θ'), topology L and error so far δ' as inputs and returns updated error, weights and bias (line 8). As the BP is derived by assuming that it is desirable to minimise the error on the output neurons over all the samples presented to NN, the error δ' is calculated as an

TABLE 2. Notation used in Algorithms 5, 6 and 7.

Symbol	Definition
$r'_{i,j}$	The encrypted values of the jth attribute in the ith record of the encrypted training set D'.
$t'_{i,j}$	The encrypted target class value of the jth neuron in the output layer for the ith encrypted training sample $r_i \in D'$.
y'^{j}_{i}	The encrypted output of the ith neuron in jth layer.
θ'^{j}_{i}	The encrypted bias value of the ith neuron in jth layer.
$w'^{i}_{x\ y}$	The encrypted weight connecting the xth neuron in the ith layer with yth neuron in the following layer (i+1).
δ'^{j}_{i}	The encrypted error value corresponding to ith neuron in jth layer.
$\Delta w'^{i}_{x\ y}$	The encrypted value of change in weight that connects xth neuron in ith layer with the yth neuron in the following layer (i+1).
$\Delta\theta'^{j}_{i}$	The encrypted value of change in bias of ith neuron in jth layer.

average overall sample (equation in line 9). The overall error will then be compared with the threshold ϵ' in line 10.

The PPFF process is given in Algorithm 6, the inputs are current encrypted data sample r', network topology L, degree of polynomial to approximate sigmoid d, set of encrypted weights W' and biases Θ'. In the PPFF, the input data sample r' is applied to the input layer and its effect is propagated, layer by layer, through the network until an output is produced. Therefore, the output of input layer neurons (y'^{1}_{i} $\forall$ *neuron* $i \in l_1$) is matched to the attribute values in a current training sample r' (lines 2 and 3); recall the l_1 in line 2 is the number of attributes in the attribute set (the number of values in each record and thus the number of neurons in the input layer). The remaining layers, the hidden layers and the output layer, the input of each neuron is calculated as the weighted sum by multiplying the output of neurons in the previous layer with the weights connecting the two layers of neurons (lines 4 to 6). To decide whether a neuron fires or not, the input is passed onto an appropriate activation function. In PPFF, the TaylorLinear approximation of the sigmoid function with degree d, φ^d given in Section V, is used. The result from φ^d determines the neuron output that becomes the input for the neuron in the next connected to it (line 7). The PPFF calculations were performed using additive and multiplicative MLS properties. To manage the growth of cyphertexts dimensionality reduction was performed after each multiplication. The PPFF algorithm will exit with a set of neuron output Y' in line 8.

The PPBP is given in Algorithm 7. At the PPBP stage, the network weights and biases are adjusted to minimise the error function. With respect to the PPNNBP the BP used the pattern mode, or what is also sometimes referred to as the online method, where the weights and bias updates are applied after the presentation of each training sample. The inputs are the set of current neurons output Y', expected class label t'_s, network topology L, the overall error value δ' and set of encrypted weights W' and biases Θ'. The algorithm starts by defining the variable one$'$ that will be initialised with the MLS encrypted equivalent of 1 (line 2). A loop is then commenced with calculating the error for each neuron in the output layer (layer b). In standard BP, the error for neuron i in the output layer is calculated by comparing the expected output with the actual network output value as per (4) where $y_i(1 - y_i)$ is the derivative of the sigmoid function. This is computed in a secure manner in lines 3 to 5 over encrypted data using MLS properties. Since all the hidden neurons have, to some degree, contributed to the errors evident in the output layer, the encrypted output errors, δ'^{b}_{i}, are transmitted backwards from the output layer to each neuron in the hidden layer that immediately contributed to the output layer. This process is then repeated layer by layer until each neuron in the network has received an error that describes its relative contribution to the overall error. The errors for neurons in the hidden layers are calculated in lines 6 to 8 starting with the layer immediately preceding the output layer (the $|L| - 1$th layer).

$$\delta^{b}_{i} = (y_i - t_i) \times (y_i \times (1 - y_i)) \tag{4}$$

Once the error for each neuron has been determined, the errors are then used by the neurons to update the values for each weight and biase as per the equations in lines 11, 12, 14 and 15. As illustrated in Table 2, $\Delta\Theta'^{j}_{i}$ is the change in the bias of the neuron i in layer j, $\Delta w'^{j}_{i\,k}$ is the change in the weight between neurons i and k that connect layer j and following layer ($j+1$), and η is the learning rate. To accelerate the learning process momentum (μ) is used to encourage the changes to continue in the same direction with larger steps. As the iterative process of incremental adjustment continues, the weights and biases will gradually converge to a locally optimal set of values that minimise the loss function; in the best case scenario globally optimal values will be reached.

VII. EXPERIMENTAL EVALUATION

This section presents the evaluation and analysis of the MLS and PPNNBP using synthetic datasets and benchmark datasets taken from the UCI data repository [46]. The synthetic datasets were used to evaluate the MLS performance, whereas the UCI datasets (listed in Table 3) were used to evaluate the PPNNBP process. Both MLS and PPNNBP were implemented in the Java programming language. Java Remote Method Invocation (RMI) technology was developed to similate the client-server that both run on the same PC [47]. The PC on which the experiments were conducted was equipped with macOS High Sierra operating system, 8 GB memory and 3.8 GHz Intel Core i5 CPU. The experiments were conducted using Ten Cross Validation (TCV); the results presented in the following subsections are therefore average values. Minmax data normalization was applied to the dataset. Table 3 also lists the number of records and attributes in each dataset, the network (input-hidden-output layer) topology, the learning rate η, and momentum μ parameter settings that were used for the experimentation. The number of epochs was fixed at *maxEpoch* $= 100$ in all cases. The NN weights and biases were initialised randomly from the range $[-0.4, 0.4]$. In practice these parameter settings would be pre-defined by the data owner. The PPNNBP was trained using the TaylorLinear with $d = 3$ (φ^3). In the prediction stage the φ^3 and FriendlyFunction (ϕ) were used.

TABLE 3. Experiment datasets and neural network parameters (*n*: number of samples, *a* number of attributes, η learning rate, and μ momentum).

Dataset	n	a	Topology	η	μ
1.Banknote Auth.	1372	4	{4, 5, 2}	0.01	0.7
2. Blood Trans.	748	4	{4, 5, 2}	0.02	0.8
3. Breast Cancer	198	33	{33, 10, 2}	0.20	0.9
4. Breast Tissue	106	9	{9, 6, 6}	0.20	0.9
5. Chronic kidney	400	24	{24, 5, 2}	0.20	0.9
6. Dermatology	366	34	{34, 5, 6}	0.20	0.8
7. Ecoli	336	7	{7, 5, 8}	0.30	0.8
8. Iris	150	4	{4, 5, 3}	0.20	0.7
9. Leafs	340	15	{15, 10, 30}	0.40	0.5
10. Lenses	24	4	{4, 10, 3}	0.50	0.7
11. Libras Move.	360	90	{90, 10, 15}	0.30	0.5
12. Parkinsons	195	22	{22, 5, 2}	0.30	0.9
13. Pima Disease	768	8	{8, 5, 2}	0.20	0.9
14. Seeds	210	7	{7, 5, 3}	0.20	0.9

The objectives of the evaluation were to analyse: (i) the operation of the MLS, (ii) the PPNNBP in terms of the complexity of data owner participation, (iii) the computational overhead of PPNNBP in comparison with standard NN (PPNNBP efficiency), (iv) the effectiveness of the approach (PPNNBP accuracy), (v) the overall security, and (vi) compare PPNNBP with state-of-the-art frameworks. Each is discussed in further detail in the following six subsections, Subsections VII-A to VII-F.

A. MLS PERFORMANCE EVALUATION

In this subsection the MLS evaluation is presented. MLS was evaluated by analysing the performance of the various supported MLS operations. Performance was measured in terms of the runtime required to: (i) generate the MLS key, (ii) encrypt data, (iii) decrypt data, (iv) utilise the FHE mathematical properties ($\oplus$,$\bigotimes$,$\circledast$), and (v) secure comparison. In the experiments the number of subcyphertexts (m) considered was $m = \{3, 9, 15\}$. The recorded runtimes to generate the MLS key were 1.16 ms, 1.37 ms and 1.44 ms for $m = 3, 9$ and 15 respectively. These results demonstrated that the runtimes for generating the MLS keys increased with the number of subcyphertexts m, the number of elements in the secret key list SK(m), this was to be expected.

The performance associated with MLS encryption and decryption, the HE mathematical properties ($\oplus$,$\bigotimes$ and $\circledast$) and order preserving properties, were also measured in terms of the required runtime to perform the operations in the context of different sizes of data records and the different numbers of subcyphertexts featured in MLS. The results are presented in Fig. 2. The data encryption, decryption and the homomorphic operations featured "linear" processing time in relation to the size of the data and the number of subcyphertexts m. However, the runtimes were negligible; using MLS a record with 1,000 attributes can be encrypted in 0.85 ms when $m = 15$, and decrypted in 0.52 ms. The HE mathematical properties ($\oplus$,$\bigotimes$ and $\circledast$) were more expensive, in terms of runtime, than encryption and decryption although the multiplication ($\bigotimes$) runtime was much higher than the addition because of dimensionality reduction. The runtime associated with the HE mathematical operations increased with the number of attributes featured in the data, and the number of subcyphertexts in the MLS. However, the times reported, as shown in Fig. 2, were again negligible. The secure comparison of two MLS cyphertexts can be achieved by comparing the qth subcyphertexts, therefore, regardless of the number of subcyphertexts (the value of m) the recorded data comparison runtime was constant at 0.2 ms.

B. DATA OWNER PARTICIPATION

This subsection considers the amount of data owner participation required to: (i) prepare the data prior to network generation, (ii) train the network using the PPNNBP framework, and (iii) predict class label using PaaS. The results for data owner preparations are presented in Table 4. The data preparation comprised: Minmax data normalization (column 2); data encryption, excluding MLS key generation and trapdoor calculation (column 3); and preparation of the training and testing samples to facilitate stratified CV (column 4). Inspection of the table shows that the data owner participation in preparing the data for network generation was negligible, and did not introduce any overhead on behalf of the data owner. The largest dataset, in terms of number of samples and attributes, "Libras Move" only required, on average, 3.86 ms for data normalisation, 3.84 ms for encryption and 6.98 ms for stratified CV data preparation.

The data owner participation with respect to network training is given in column 8 of Table 4, measured in terms of the average runtime over all ten of the TCV folds. Recall that data owner involvement in the model training is limited to division (inversion) operations with respect to the TaylorLinear approximation of the sigmoid activation function (whenever it is encountered). It is possible to implement PaaS using two different methods, TaylorLinear and FriendlyFunction, which differ in how activation functions are approximated and how much data owners are involved. TaylorLinear approximation requires data owner participation. As in the case of training the model; the time complexity for data owner participation using TaylorLinear will be in the order of $O(\sum_{i=1}^{i=b} l_i)$; the number of neurons in NN. Data owners will decrypt approximate activation function values, reverse the values, encrypt the results, and send them to the cloud. The data owner participation for PaaS was evaluated using the "Libras Move" dataset because this was associated with the largest number of neurons in the generated network (see Table 3). Predicting the label for one query record in the "Libras Move" dataset, using a network topology of {90; 10; 15} was 0.14 ms. Note that FriendlyFunction approximation can be entirely conducted using the homomorphic operations facilitated by the

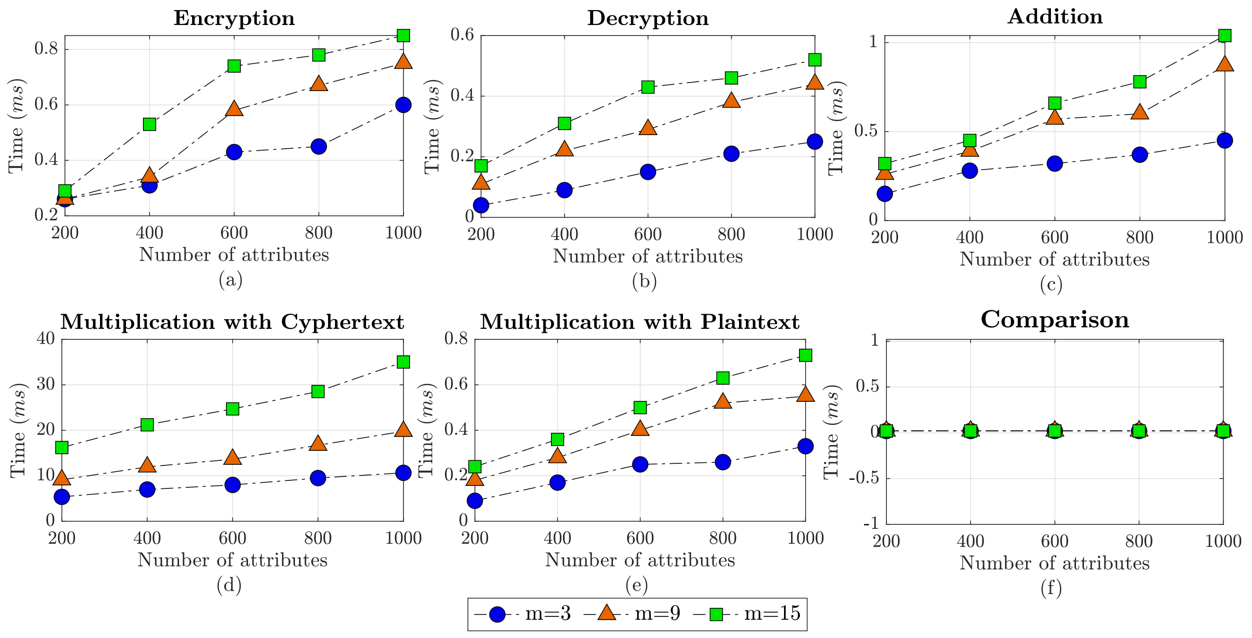

FIGURE 2. The MLS performance evaluation for different value of *m* and different numbers of attributes in a record. The runtime values were averaged over 10 folds of TCV.

MLS properties, therefore no data owner participation was required.

C. PPNNBP EFFICIENCY

The total runtime for training each network using PPNNBP is given in column 6 of Table 4. Column 5 gives the runtime to train the same network without using any encryption. Note that the runtimes given in column 6 are in seconds (s), whilst those given in column 5 are in milliseconds (ms). As expected, training a NN over encrypted data introduces a computational overhead. The difference is due to the computation complexity of the FHE mathematical properties and the linear approximation of the sigmoid function using TaylorLinear. However, it is argued here, that this is not an unacceptable overhead, even for the largest dataset, the "Libras Move" dataset, the network was trained in 873.85 s.

Further experiments were conducted using a single machine to investigate the computational overhead associated with using a trained PPNNBP to provide PaaS, as compared to standard NN (over plaintext neural parameters). The results indicated that the runtime was negligible. The "Libras Move" dataset was again considered in this respect as it had the largest number of neurons in the NN topology and the largest number of class labels (see Table 3). Using TaylorLinear approximation, where $d = 3$, 1, 641, 256 predictions could be made per hour. In contrast, when using FriendlyFunction approximation 4, 143, 012 predictions could be made per hour. The use of standard NN coupled with the standard sigmoid function allows 655, 463, 103 predictions to be made every hour. Therefore, it can be concluded that the standard NN is more efficient than the PPNNBP using TaylorLinear and FriendlyFunction, although FriendlyFunction is more efficient than the linear approximation using TaylorLinear.

D. PPNNBP ACCURACY

The classification accuracy obtained using standard NN over unseen data samples was compared with the accuracy of the PPNNBP approach using both TaylorLinear and FriendlyFunction approximation and the same network topology and parameters. The intuition was that the PPNNBP should produce comparable results to those obtained using the standard NN; if so the PPNNBP could be said to be operating correctly. The accuracy evaluation metrics were: (i) Precision (P), Recall (R), the F1 measure and accuracy (Acc) [48], and (ii) the value of the loss function calculated for different numbers of epochs in TaaS. To provide a precise and fair comparison, the performance measures were calculated over the same test set for all activation functions (sigmoid, φ, and ϕ). This approach was previously used in [41] and [49] for comparing performances of different activation functions.

Table 5 shows the P, R, F1, and Acc values obtained when using the standard and PPNNBP frameworks. From the table it can be seen that:

P: For ten of the datasets considered the precision (P) values obtained for all three approaches were more-or-less equal. For the remaining four cases, "Blood Trans", "Leafs", "Lenses", and "Libras Move", the values obtained using TaylorLinear were comparable with the standard approach and equal to it in "Lenses"; whilst using Friendly Function the precision values obtained were slightly lower.

R: Recall (R) values were similar in nine cases. In the remaining five cases, two cases, "Breast Tissue" and "Libras Move", the TaylorLinear produced comparable results to sigmoid, whilst FriendlyFunction produced slightly lower values.

TABLE 4. Runtime for data owner data preparation and network training operating statistics.

Dataset	Data Preparation (ms)	Data Encryption (ms)	CV time (ms)	Stand.NN Execution time (ms)	PPNNBP		
					Total execution time (s)	Data miner process (s)	Data owner process (s)
1. Banknote Auth.	1.08	1.59	5.63	632.64	487.13	249.49	237.64
2. Blood Trans.	1.10	1.08	1.95	365.82	283.69	155.21	128.49
3. Breast Cancer	2.21	2.80	1.96	268.58	195.32	132.67	62.65
4. Breast Tissue	1.20	0.47	0.64	124.32	67.25	36.41	30.84
5. Chronic kidney	1.51	2.17	3.95	275.72	189.05	118.36	70.69
6. Dermatology	1.75	2.58	3.92	375.04	263.81	172.58	91.23
7. Ecoli	0.74	0.95	1.84	313.13	240.77	141.09	99.69
8. Iris	0.52	0.28	0.64	101.71	65.69	34.45	31.24
9. Leafs	0.92	1.72	3.29	104.39	83.30	5.53	77.77
10. Lenses	0.49	0.08	0.11	4.19	1.82	0.14	1.68
11. Libras Move.	3.86	3.84	6.98	1202.13	873.85	656.75	217.11
12. Parkinsons	1.93	0.84	1.35	159.41	91.50	58.73	32.77
13. Pima Disease	1.79	3.00	3.04	410.20	312.84	178.40	134.44
14. Seeds	0.75	0.96	1.02	141.08	94.69	53.22	41.47

The "Lenses" produced the same result as sigmoid whilst the result was comparable using Friendly-Function. In the case of the "Blood Trans" dataset, the recall values obtained using TaylorLinear and the sigmoid function were identical, however the value obtained using FriendlyFunction was slightly higher. In the case of the "Leafs, FriendlyFunction is lower than sigmoid and TaylorLinear.

F1: With respect to the F1 values obtained, these were comparable in ten cases; whereas in one case, "Breast Tissue", TaylorLinear and FriendlyFunction produced identical values slightly lower than the sigmoid function. In the remaining cases, "Leafs", "Lenses", and "Libras Move", the TaylorLinear was equal to sigmoid function in the case of "Lenses", higher in the case of "Libras Move" and comparable to the sigmoid function in "Leafs" whilst FriendlyFunction was slightly lower in all cases.

Acc: In terms of accuracy (Acc), in six datasets the values obtained for the TaylorLinear approach were exactly the same as the sigmoid approach. Three of these cases were also same as FriendlyFunction approach; "Banknote Auth.", "Breast Cancer", and "Chronic Kidney". For "Dermatology", "Iris", "Libras Move", and "Parkinsons" the Acc obtained using TaylorLinear were slightly higher than the sigmoid approach whilst FriendlyFunction produced the same Acc as sigmoid in the case of "Dermatology" and "Iris" and less than sigmoid in the remaining cases. In "Breadth Tissue", "Ecoli", and "Leafs", TaylorLinear obtained comparable Acc to sigmoid and less to the FriendlyFunction. In "Pima Disease" TaylorLinear and FriendlyFunction produced same results lower than the sigmoid approach.

The overall average values for precision were 0.81, 0.80 and 0.77 for standard NN using the sigmoid function, PPNNBP using TaylorLinear and PPNNBP using Friendly-Function respectively. The overall average values for recall were 0.78, 0.78, and 0.75 respectively. The average F1 values were 0.79, 0.78, and 0.75; and the average Acc values were 0.84, 0.84, and 0.79. Therefore, it can be concluded that the combined PPNNBP approach with TaylorLinear approximation produced comparable results to standard NN without encryption. The results produced using FriendlyFunction approximation were not as accurate, because the approximation was coarser than the values produced using TaylorLinear approximation (as shown in Fig. 1).

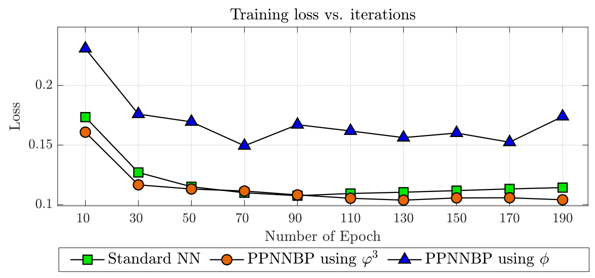

FIGURE 3. Loss function for the three NN for different number of epochs.

The loss function values, during TaaS, were also compared between standard and PPNNBP using the two different activation functions with respect to different numbers of epochs. Fig. 3 shows the loss function for standard and PPNNBP for different epochs from 10 to 190 in steps of 20. From the figure it can be seen that in all cases FriendlyFunction produced the worst performance. The operation of the PPNNBP framework coupled with TaylorLinear approximation and standard NN were comparable. Thus it was concluded that TaylorLinear approximation can approximate the value of the sigmoid function whereas maintaining the overall accuracy of the trained model.

E. SECURITY

Using the PPNNBP framework the third party, cloud provider, is considered to be an "Honest but Curious" party;

TABLE 5. Prediction accuracies using: (i) standard NN with sigmoid activation, (ii) PPNNBP with φ^3 approximation, and (iii) PPNNBP with ϕ approximation.

Dataset	Standard NN				PPNNBP							
					$TaylorLinear\ (\varphi^3)$				$FriendlyFunction\ (\phi)$			
	P	R	F1	Acc	P	R	F1	Acc	P	R	F1	Acc
1. Banknote Auth.	0.98	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99
2. Blood Trans.	0.73	0.54	0.51	0.77	0.71	0.54	0.51	0.77	0.61	0.65	0.55	0.55
3. Breast Cancer	0.65	0.65	0.66	0.75	0.65	0.65	0.66	0.75	0.66	0.67	0.67	0.75
4. Breast Tissue	0.66	0.64	0.64	0.66	0.61	0.61	0.57	0.63	0.60	0.57	0.57	0.59
5. Chronic Kidney	0.97	0.98	0.98	0.98	0.97	0.98	0.98	0.98	0.97	0.98	0.98	0.98
6. Dermatology	0.94	0.94	0.94	0.95	0.96	0.96	0.97	0.97	0.95	0.94	0.95	0.95
7. Ecoli	0.62	0.59	0.61	0.87	0.62	0.62	0.62	0.86	0.58	0.59	0.59	0.83
8. Iris	0.97	0.97	0.97	0.97	0.98	0.98	0.98	0.98	0.97	0.97	0.97	0.97
9. Leafs	0.68	0.69	0.67	0.68	0.62	0.64	0.62	0.64	0.56	0.34	0.40	0.54
10. Lenses	0.76	0.73	0.74	0.79	0.76	0.73	0.74	0.79	0.68	0.68	0.66	0.70
11. Libras Move.	0.78	0.76	0.76	0.76	0.77	0.77	0.77	0.77	0.70	0.67	0.68	0.71
12. Parkinsons	0.85	0.82	0.84	0.88	0.85	0.83	0.85	0.89	0.82	0.83	0.83	0.87
13. Pima Disease	0.75	0.72	0.74	0.77	0.75	0.70	0.72	0.76	0.75	0.71	0.72	0.76
14. Seeds	0.94	0.94	0.95	0.94	0.94	0.94	0.95	0.94	0.91	0.91	0.91	0.91
Average	**0.81**	**0.78**	**0.79**	**0.84**	**0.80**	**0.78**	**0.78**	**0.84**	**0.77**	**0.75**	**0.75**	**0.79**

thus the semi-honest security model was considered where the third party executes the stated algorithm without deviation and does not fail to provide the required service. At the same time the third party is curious, in the sense that it would "look" at the available data and calculated intermediate results. The security of the PPNNBP framework was thus evaluated in terms of the semi-honest model by identifying potential attacks that can be instigated during network training (TaaS) and provision of PaaS. Model training was performed on encrypted data, encrypted data labels, and encrypted NN weights and biases, thus the only potential form of attack was a COAs available whenever adversaries have access to such cyphertexts. The MLS, as in the case of original LS, is a probabilistic scheme that produces different cyphertexts for the same plaintext value each time it is applied, even when using the same secret key. This feature means that MLS cyphertexts are semantically secure, hence accessing cyphertexts does not provide any useful information, with respect to the associated plaintext, from the perspective of an adversary. COAs are more likely to succeed when attackers have a background knowledge of the data frequency of the original data values. Knowledge associated with the ordering feature of some order preserving encryption schemes might allow an adversary to infer the ranges containing dense data. Alternatively, frequency analysis could allow attackers to highlight cyphertexts with the same frequency as plaintexts (if such plaintexts were available) and then identify cyphertexts that have the same frequency. However, this will not be possible in the case of MLS because different cyphers are produced for the same plaintext values using the ω-concept presented in Subsection IV-A. The entire model training was conducted over MLS cyphers and no decryption took place at the cloud side which implies even more security. Hence it is argued that the PPNNBP framework, founded on MLS, is secure with respect to COAs.

In the context of PaaS offered by a cloud provider, there were two possible issues of concern in the PPNNBP framework: (i) the presence of sensitive information in prediction requests, and (ii) the knowledge embedded in the trained model might be accessible to external adversities. In the context of the first concern, prediction requests are sent in encrypted form, the third party performs the requested inference over the encrypted data and then produces an encrypted prediction that only be decrypted by the data owner. In context of the second concern, all weights and biases are encrypted using MLS which, as noted above, has semantic security features; therefore "inversion" attacks can not be instigated without access to the required encryption key.

F. COMPARISON WITH STATE-OF-ART FRAMEWORKS

In this subsection, experiments were conducted to compare the PPNNBP framework and its building blocks with existing solutions that provide TaaS and PaaS such as SecureNN [33], SecureML [37], and QUOTIENT [38]. The objective of the experiments is to compare the accuracy of model, the security provided, and the complexity of: (i) MLS multiplication, (ii) secure comparison, and (iii) model training and prediction with solutions introduced in other frameworks.

The PPNNBP training accuracy was compared with that of SecureNN (with three layers), SecureML (using ReLU and quadratic activation functions) and QUOTIENT (shown in Table 6). As a baseline evaluation of training, we compare secure training performance to its standard (unencrypted) counterpart. In order to provide an accurate comparison, accuracy loss is used. This is calculated as a difference between the secure framework accuracy and the accuracy of the standard model. As shown in Table 6, all frameworks achieved accuracy levels similar to the standard approach; all accuracy losses were less than 0.014. It is only in the PPNNBP approach that secure training accuracy is higher than that of the standard approach, and therefore the accuracy loss is positive. This was due to the approximation used to calculate the activation function.

For the purpose of security, training data, intermediate results, and NN weights and biases must be preserved. In PPNNBP the training data were encrypted using a key

TABLE 6. Comparing the accuracy of TaaS using different frameworks to standard (unencrypted) equivalents.

Framework	Training Accuracy	Accuracy loss
SecureNN (3 Layers)	0.934	-0.006
Standard NN	0.940	
SecureML (ReLU)	0.934	-0.011
SecureML (Quadratic)	0.931	-0.014
Standard NN	0.945	
QUOTIENT	0.897	-0.011
Standard NN	0.908	
PPNNBP (φ^3)	0.928	+0.008
Standard NN	0.920	

TABLE 7. Comparison of our MLS based multiplication of k-dimensional vectors with activation function values with QUOTIENT using OT and GC and SecureML using OT and LHE.

k	QUOTIENT		SecureML		PPNNBP
	OT (s)	GC (s)	OT (s)	LHE (s)	MLS with m=3 (s)
10^3	0.08	0.025	0.028	5.3	0.002
10^4	0.08	0.140	0.160	53.0	0.005
10^5	0.13	1.410	1.400	512.0	0.036
10^6	0.60	13.120	14.000	5000.0	0.252
10^7	6.00	139.800	140.000	50000.0	1.314

belonging to the data owner that was not disclosed to the server. Intermediate results were also protected and not disclosed to the server as only cyphertexts were manipulated. However, comparable TaaS and PaaS frameworks rely on schemes and protocols that secretly share training data and intermediate results across two or three non-colluding servers (as shown in Table 1). Therefore, the security guarantees provided by these frameworks are weaker than those provided by PPNNBP, as data privacy may be compromised when servers collude. In PPNNBP, the only information disclosed to the server is the NN topology, which is essential for PPNNBP to operate correctly. This was also the case in SecureNN [33], SecureML [37], and QUOTIENT [38] frameworks.

MLS introduces the subcyphertexts dimensionality reduction mechanism to manage the size of cyphertexts throughout the training process. Table 7 compares the runtimes of MLS homomorphic multiplications, followed by dimensionality reduction $\bigotimes$, with the approach facilitated by the OT and Garbled Circuit (GC) introduced in the QUOTIENT framework and the OT and LHE based approaches introduced in SecureML. The runtime measures the time required to multiply a vector with k-dimensions by the value of an activation function. The experiments demonstrate that MLS based multiplication outperforms all other approaches. Thus, MLS is well suited to applications requiring extensive multiplication, such as DNN.

Experiments show that data comparisons using MLS property take 0.2 ms and run on cloud without data owner participation (Fig. 2). The cloud will know the comparison result without revealing the exact value of cyphertexts. Comparable solutions such as the PrivateCompare protocol introduced in [33], performs data comparison in a bit-wise manner and require the involvement of three parties. Therefore, the number of communications required depends on the number of bits in the data to be compared. As an example, values with l-bit with a maximum number of bits of p will have complexity equal to $2 \times l \times log(p)$. In addition, at the end of the PrivateCompare protocol, the server that runs the training and prediction algorithms will be able to obtain the results of the data comparison in addition to the exact value of the most significant bits.

The experiments conducted in [33] and [38] demonstrate that SecureNN and QUOTIENT frameworks require long training times even for small datasets. As an example, in [38], the reported training time for NN using the "breast cancer" dataset was 14.51 h, while for PPNNBP, the training time was 195.32 s. These are clearly the result of the complexity of the adopted SMPC protocols. In the case of SecureNN, the prediction time was 0.045 s, which means that 90, 000 predictions can be made in an hour as opposed to 1, 641, 256 using TaylorLinear and 4, 143, 012 using FriendlyFunction approximation. Results indicate that PPNNBP is more efficient than comparable frameworks.

There are potential limitations to the PPNNBP. The framework was evaluated on a single computer without a network simulation which measured the potential network overhead associated with the involvement of data owners. In PPNNBP, the involvement of data owners are expected to be small. For example, in TaaS this is limited to $O(n * \sum_{i=1}^{i=b} l_i)$ and for PaaS it is limited to $O(\sum_{i=1}^{i=b} l_i)$ where n is number of samples in training data, b is number of layers specified in NN topology, and l_i is number of neurons in the ith layer. This is not like using the SMPC protocols that required extensive communication between multiple parties.

VIII. CONCLUSION AND FUTURE WORK

In this paper the PPNNBP framework, supported by MLS, has been proposed. The framework, coupled with MLS, allows for privacy-preserving multilayer NN with BP learning. The model is trained using encrypted data and encrypted network weights and biases. It can also be used to provide secure PaaS. The training of networks and their usage does not entail any significant computational overhead over data owner, whist at the same time effectiveness is comparable with that obtained using standard NN. Thus, the framework is well suited for Secure MLaaS, which delegate TaaS and PaaS to a third party data miner with limited data owner involvement. In PPNNBP, only one cloud server is used to provide TaaS and PaaS. The advantages offered by the PPNNBP framework result from the MLS which addresses the problem, found in existing FHE schemes, of the exponential increase in cyphertext size and the inclusion of noise every time a multiplication operation is conducted. In addition, MLS incorporates a MLS feature, the ω-concept, that preserves data ordering and thus allows for secure data comparison. Using MLS, as in the case of FHE schemes, the nonlinear sigmoid activation can not be easily implemented, hence the paper proposes the TaylorLinear approximation for model training which requires some data owner participation. For providing PaaS two alternative linear approximations to the sigmoid function were considered, TaylorLinear and FriendlyFunction. The second offers the advantage that it requires no data owner participation, but at the expense of some reduction in prediction accuracy. For future work, the authors intend to investigate the utility of MLS with respect to different ML algorithms.

APPENDIX A MLS PROOF OF CORRECTNESS

A. DIMENSIONALITY REDUCTION ALGORITHM

Assuming MLS number of subcyphertext $m = 3$ and $q = 1$ (this means only $t_1 \neq 0$). The subcyphertexts for encrypting v_1 and v_2, using Algorithm 2, are calculated as follows:

$$\begin{aligned}
Encrypt(v_1, \mathrm{SK}(3)) = E_1 &= \{e_{1_1}, e_{1_2}, e_{1_3}\}\ where\\
e_{1_1} &= \frac{k_1t_1v_1 + s_1 + k_1(r_1 - r_2)}{s_1}\\
&= \frac{k_1t_1v_1 + s_1 + k_1r_1 - k_1r_2}{s_1}\\
e_{1_2} &= \frac{k_2t_2v_1 + s_2 + k_2(r_2 - r_1)}{s_2}\\
&= \frac{s_2 + k_2r_2 - k_2r_1}{s_2}\\
e_{1_3} &= (k_3 + s_3 + t_3) = kst \qquad (5)
\end{aligned}$$

$$\begin{aligned}
Encrypt(v_2, \mathrm{SK}(3)) = E_2 &= \{e_{2_1}, e_{2_2}, e_{2_3}\}\ where\\
e_{2_1} &= \frac{\cdot}{k_1t_1v_2 + s_1 + k_1(r_1 - r_2)} s_1\\
&= \frac{k_1t_1v_2 + s_1 + k_1r_1 - k_1r_2}{s_1}\\
e_{2_2} &= \frac{k_2t_2v_2 + s_2 + k_2(r_2 - r_1)}{s_2}\\
&= \frac{s_2 + k_2r_2 - k_2r_1}{s_2}\\
e_{2_3} &= (k_3 + s_3 + t_3) = kst \qquad (6)
\end{aligned}$$

Multiplying $Encrypt(v_1$,SK(3)) and $Encrypt(v_2$,SK(3)) gives $E_1 \otimes E_2 = E = \{e_{1_1}e_{2_1}, e_{1_1}e_{2_2}, e_{1_1}e_{2_3}, e_{1_2}e_{2_1}, e_{1_2}e_{2_2}, e_{1_2}e_{2_3}, e_{1_3}e_{2_1}, e_{1_3}e_{2_2}, e_{1_3}e_{2_3}\}$. Using the dimensionality reduction algorithm (Algorithm 4) the resulted cypher E can be re-encrypted without being first decrypted whilst reducing the number of subcyphertexts to only 3 subcyphertexts; when decrypted this should give $v_1 \times v_2$. Following the steps outlined in Algorithm 4, every three consecutive subcyphers in E will be associated with a trapdoor in Trap and kst which will be used to calculate one cypher in the reduced cypher RE= {re$_1$, re$_2$, re$_3$}. For example, re$_1$ is calculated from $\{e_{1_1}e_{2_1}, e_{1_1}e_{2_2}, e_{1_1}e_{2_3}\}$. Applying Algorithm 4 the reduced subcyphers will be as follows:

$$\begin{aligned}
re_1 &= (((e_{1_1}e_{2_1}) - (\frac{e_{1_1}e_{2_3}}{kst})) \times trap_1) + (((e_{1_1}e_{2_2})\\
&\quad - (\frac{e_{1_1}e_{2_3}}{kst})) \times trap_2)\\
re_2 &= (((e_{1_2}e_{2_1}) - (\frac{e_{1_2}e_{2_3}}{kst})) \times trap_1) + (((e_{1_2}e_{2_2})\\
&\quad - (\frac{e_{1_2}e_{2_3}}{kst})) \times trap_2)\\
re_3 &= (((e_{1_3}e_{2_1}) - (\frac{e_{1_3}e_{2_3}}{kst})) \times trap_1) + (((e_{1_3}e_{2_2})\\
&\quad - (\frac{e_{1_3}e_{2_3}}{kst})) \times trap_2)
\end{aligned}$$

According to Trapdoor calculation algorithm (Algorithm 1) the values for trap$_1$ and trap$_2$ are $\frac{s_1 \times secretK}{secretS \times k_1}$ and $\frac{s_2 \times secretK}{secretS \times k_2}$; and the value for kst will be $k_3 + s_3 + t_3$. After calling cypher reduction, the level of RE is one (RE.l = 1). The RE cypher can then be decrypted using Algorithm 3 to give $v_1 \times v_2$. Following Algorithm 3, as the cyphertext level value in RE is not equal to zero, a new subcyphertext value for each subcypher re$_i$ in RE is calculated (lines 3 to 5) the new RE subcyphertexts are:

$$\begin{aligned}
re_1 &= \frac{1}{t}[\frac{s_1}{k_1}(e_{1_1}e_{2_1} - \frac{e_{1_1}e_{2_3}}{kst}) + \frac{s_2}{k_2}(e_{1_1}e_{2_2} - \frac{e_{1_1}e_{2_3}}{kst})]\\
re_2 &= \frac{1}{t}[\frac{s_1}{k_1}(e_{1_2}e_{2_1} - \frac{e_{1_2}e_{2_3}}{kst}) + \frac{s_2}{k_2}(e_{1_2}e_{2_2} - \frac{e_{1_2}e_{2_3}}{kst})]\\
re_3 &= \frac{1}{t}[\frac{s_1}{k_1}(e_{1_3}e_{2_1} - \frac{e_{1_3}e_{2_3}}{kst}) + \frac{s_2}{k_2}(e_{1_3}e_{2_2} - \frac{e_{1_3}e_{2_3}}{kst})]
\end{aligned}$$

The cyphers are then used, in lines 6 and 7 of Algorithm 3, and processed the follows:

$$\begin{aligned}
t &= t_1 + t_2 = t_1 + 0 = t_1\\
s &= \frac{re_3}{(k_3 + s_3 + t_3)} = \frac{re_3}{kst}\\
v &= \frac{\frac{((re_1 \times s_1) - (\frac{re_3}{kst} \times s_1))}{k1} + \frac{((re_2 \times s_2) - (\frac{re_3}{kst} \times s_2))}{k2}}{t}\\
&= \frac{1}{t}[\frac{s_1}{k_1} \times (re_1 - \mathbf{\frac{re_3}{kst}}) + \frac{s_2}{k_2} \times (re_2 - \mathbf{\frac{re_3}{kst}})] \qquad (7)
\end{aligned}$$

The values for $\frac{re_3}{kst}$, re$_1$, and re$_2$ must then be calculated.

The values for $\frac{re_3}{kst}$ are given by:

$$\begin{aligned}
\frac{re_3}{kst} &= \frac{1}{t}[\frac{s_1}{k_1kst}(e_{1_3}e_{2_1} - \frac{e_{1_3}e_{2_3}}{kst}) + \frac{s_2}{k_2kst}(e_{1_3}e_{2_2} - \frac{e_{1_3}e_{2_3}}{kst})]\\
&= \frac{1}{t}[\frac{s_1e_{1_3}e_{2_1}}{k_1kst} - \frac{s_1e_{1_3}e_{2_3}}{k_1kst^2} + \frac{s_2e_{1_3}e_{2_2}}{k_2kst} - \frac{s_2e_{1_3}e_{2_3}}{k_2kst^2}]
\end{aligned}$$

where:

$$\begin{aligned}
\frac{s_1e_{1_3}e_{2_1}}{k_1kst} &= \frac{s_1}{k_1kst}(kst)(\frac{k_1t_1v_2 + s_1 + k_1r_1 - k_1r_2}{s_1})\\
&\quad (5)\ and\ (6)\\
&= \frac{k_1t_1v_2 + s_1 + k_1r_1 - k_1r_2}{k_1} = t_1v_2 + \frac{s_1}{k_1} + r_1 - r_2\\
\frac{s_1e_{1_3}e_{2_3}}{k_1kst^2} &= \frac{s_1}{k_1kst^2}kst\ kst = \frac{s_1}{k_1}\\
\frac{s_2e_{1_3}e_{2_2}}{k_2kst} &= \frac{s_2}{k_2kst}e_{1_3}e_{2_2} = \frac{s_2}{k_2kst}kst(\frac{s_2 + k_2r_2 - k_2r_1}{s_2})\\
&= \frac{s_2 + k_2r_2 - k_2r_1}{k_2} = \frac{s_2}{k_2} + r_2 - r_1\\
\frac{s_2e_{1_3}e_{2_3}}{k_2kst^2} &= \frac{s_2}{k_2kst^2}kst\ kst = \frac{s_2}{k_2}
\end{aligned}$$

In other words:

$$\frac{re_3}{kst} = \frac{1}{t}[t_1v_2 + \frac{s_1}{k_1} + r_1 - r_2 - \frac{s_1}{k_1} + \frac{s_2}{k_2} + r_2 - r_1 - \frac{s_2}{k_2}]$$
$$= \frac{1}{t}t_1v_2 = v_2 \quad given\ that\ t = t_1$$

The values for re_1 and re_2 (in (7)) are given by:

$$re_1 = \frac{1}{t}[\frac{s_1e_{1_1}e_{2_1}}{k_1} - \frac{s_1e_{1_1}e_{2_3}}{k_1kst} + \frac{s_2e_{1_1}e_{2_2}}{k_2} - \frac{s_2e_{1_1}e_{2_3}}{k_2kst}]$$

The operands in the above are calculated, retrospectively, as follows $\frac{s_1e_{1_1}e_{2_1}}{k_1}$, shown at the top of the next page.

In other words (recall $t = t_1$):

$$re_1 = \frac{1}{t}[\frac{k_1t_1^2v_1v_2}{s_1} + t_1v_1 + \frac{k_1t_1r_1v_1}{s_1} - \frac{k_1t_1r_2v_1}{s_1} + t_1v_2$$
$$+ \frac{s_1}{k_1} + 2\,r_1 - 2r_2 + \frac{k_1t_1r_1v_2}{s_1} + \frac{k_1r_1^2}{s_1} - 2\frac{k_1r_1r_2}{s_1}$$
$$- \frac{k_1t_1r_2v_2}{s_1} + \frac{k_1r_2^2}{s_1} - t_1v_1 - \frac{s_1}{k_1} - r_1 + r_2$$
$$+ \frac{k_1t_1s_2v_1}{s_1k_2} + \frac{k_1t_1r_2v_1}{s_1} - \frac{k_1t_1r_1v_1}{s_1} + \frac{s_2}{k_2} + r_2$$
$$- r_1 + \frac{k_1r_1s_2}{s_1k_2} + 2\frac{k_1r_1r_2}{s_1} - \frac{k_1r_1^2}{s_1} - \frac{k_1r_2s_2}{s_1k_2}$$
$$- \frac{k_1r_2^2}{s_1} - \frac{k_1t_1s_2v_1}{s_1k_2} - \frac{s_2}{k_2} - \frac{k_1s_2r_1}{s_1k_2} + \frac{k_1s_2r_2}{s_1k_2}]$$
$$= \frac{k_1t_1v_1v_2}{s_1} + v_2 + \frac{k_1r_1v_2}{s_1} - \frac{k_1r_2v_2}{s_1} \tag{8}$$

Recall that $t = t_1 + t_2$, however, as the key generation conditions require that there is only one $t_q \neq 0$ that is t_1 thus $t = t_1$.

The value for re_2 (in (7)) is then given by:

$$re_2 = \frac{1}{t}[t_1v_2 + \frac{s_1}{k_1} + r_1 - r_2 + \frac{t_1k_2r_2v_2}{s_2} + \frac{s_1k_2r_2}{k_1s_2}$$
$$+ 2\frac{k_2r_1r_2}{s_2} - \frac{k_2r_2^2}{s_2} - \frac{k_2t_1r_1v_2}{s_2} - \frac{s_1k_2r_1}{k_1s_2} - \frac{k_2r_1^2}{s_2}$$
$$- \frac{s_1}{k_1} - \frac{s_1k_2r_2}{k_1s_2} + \frac{s_1k_2r_1}{k_1s_2} + \frac{s_2}{k_2} + 2\,r_2 - 2\,r_1 +$$
$$\frac{k_2r_2^2}{s_2} - 2\frac{k_2r_1r_2}{s_2} + \frac{k_2r_1^2}{s_2} - \frac{s_2}{k_2} - r_2 + r_1]$$
$$= \frac{1}{t}[t_1v_2 + \frac{t_1k_2r_2v_2}{s_2} - \frac{k_2t_1r_1v_2}{s_2}]$$
$$= v_2 + \frac{k_2r_2v_2}{s_2} - \frac{k_2r_1v_2}{s_2}$$

where:

$$\frac{s_1e_{1_2}e_{2_1}}{k_1} = \frac{s_1}{k_1}[(\frac{s_2 + k_2r_2 - k_2r_1}{s_2})$$
$$(\frac{k_1t_1v_2 + s_1 + k_1r_1 - k_1r_2}{s_1})]$$
$$= \frac{s_1}{k_1}[(1 + \frac{k_2r_2}{s_2} - \frac{k_2r_1}{s_2})(\frac{k_1t_1v_2}{s_1} + 1 + \frac{k_1r_1}{s_1}$$
$$- \frac{k_1r_2}{s_1})]$$
$$= \frac{s_1}{k_1}[\frac{k_1t_1v_2}{s_1} + 1 + \frac{k_1r_1}{s_1} - \frac{k_1r_2}{s_1} + \frac{t_1k_1k_2r_2v_2}{s_1s_2}$$
$$+ \frac{k_2r_2}{s_2} + \frac{k_1k_2r_1r_2}{s_1s_2} - \frac{k_1k_2r_2^2}{s_1s_2} - \frac{k_1k_2t_1r_1v_2}{s_1s_2}$$
$$- \frac{k_2r_1}{s_2} - \frac{k_1k_2r_1^2}{s_1s_2} + \frac{k_1k_2r_1r_2}{s_1s_2}]$$
$$= t_1v_2 + \frac{s_1}{k_1} + r_1 - r_2 + \frac{t_1k_2r_2v_2}{s_2} + \frac{s_1k_2r_2}{k_1s_2}$$
$$+ 2\frac{k_2r_1r_2}{s_2} - \frac{k_2r_2^2}{s_2} - \frac{k_2t_1r_1v_2}{s_2} - \frac{s_1k_2r_1}{k_1s_2}$$
$$- \frac{k_2r_1^2}{s_2}$$

$$\frac{s_1e_{1_2}e_{2_3}}{k_1kst} = \frac{s_1}{k_1kst}(\frac{s_2 + k_2r_2 - k_2r_1}{s_2})(kst)$$
$$= \frac{s_1s_2 + s_1k_2r_2 - s_1k_2r_1}{k_1s_2}$$
$$= \frac{s_1}{k_1} + \frac{s_1k_2r_2}{k_1s_2} - \frac{s_1k_2r_1}{k_1s_2}$$

$$\frac{s_2e_{1_2}e_{2_2}}{k_2} = \frac{s_2}{k_2}(\frac{s_2 + k_2r_2 - k_2r_1}{s_2})(\frac{s_2 + k_2r_2 - k_2r_1}{s_2})$$
$$= \frac{s_2}{k_2}(1 + \frac{k_2r_2}{s_2} - \frac{k_2r_1}{s_2})(1 + \frac{k_2r_2}{s_2} - \frac{k_2r_1}{s_2})$$
$$= \frac{s_2}{k_2}[1 + \frac{k_2r_2}{s_2} - \frac{k_2r_1}{s_2} + \frac{k_2r_2}{s_2} + \frac{k_2^2r_2^2}{s_2^2} - \frac{k_2^2r_1r_2}{s_2^2}$$
$$- \frac{k_2r_1}{s_2} - \frac{k_2^2r_1r_2}{s_2^2} + \frac{k_2^2r_1^2}{s_2^2}]$$
$$= [\frac{s_2}{k_2} + r_2 - r_1 + r_2 + \frac{k_2r_2^2}{s_2} - \frac{k_2r_1r_2}{s_2} - r_1$$
$$- \frac{k_2r_1r_2}{s_2} + \frac{k_2r_1^2}{s_2}]$$
$$= [\frac{s_2}{k_2} + 2\,r_2 - 2\,r_1 + \frac{k_2r_2^2}{s_2} - 2\frac{k_2r_1r_2}{s_2} + \frac{k_2r_1^2}{s_2}]$$

$$\frac{s_2e_{1_2}e_{2_3}}{k_2kst} = \frac{s_2}{k_2kst}(\frac{s_2 + k_2r_2 - k_2r_1}{s_2})(kst) = \frac{s_2}{k_2} + r_2 - r_1$$

Finally:

$$v = \frac{1}{t}[\frac{s_1}{k_1} \times (\frac{k_1t_1v_1v_2}{s_1} + v_2 + \frac{k_1r_1v_2}{s_1} - \frac{k_1r_2v_2}{s_1} - v_2)$$
$$+ \frac{s_2}{k_2} \times (v_2 + \frac{k_2r_2v_2}{s_2} - \frac{k_2r_1v_2}{s_2} - v_2)]$$
$$= \frac{1}{t}[t_1v_1v_2 + r_1v_2 - r_2v_2 + r_2v_2 - r_1v_2]$$
$$= \frac{1}{t}[t_1v_1v_2]$$
$$= v_1v_2$$

As expected from decrypting cyphertext RE.

$$
\begin{aligned}
\frac{s_1 e_{1_1} e_{2_1}}{k_1} &= \frac{s_1}{k_1}[(\frac{k_1t_1v_1 + s_1 + k_1r_1 - k_1r_2}{s_1})(\frac{k_1t_1v_2 + s_1 + k_1r_1 - k_1r_2}{s_1})] \\
&= \frac{s_1}{k_1}[(\frac{k_1t_1v_1}{s_1} + 1 + \frac{k_1r_1}{s_1} - \frac{k_1r_2}{s_1})(\frac{k_1t_1v_2}{s_1} + 1 + \frac{k_1r_1}{s_1} - \frac{k_1r_2}{s_1})] \\
&= \frac{s_1}{k_1}[\frac{k_1^2t_1^2v_1v_2}{s_1^2} + \frac{k_1t_1v_1}{s_1} + \frac{k_1^2t_1r_1v_1}{s_1^2} - \frac{k_1^2t_1r_2v_1}{s_1^2} + \\
&\quad \frac{k_1t_1v_2}{s_1} + 1 + \frac{k_1r_1}{s_1} - \frac{k_1r_2}{s_1} + \frac{k_1^2t_1r_1v_2}{s_1^2} + \frac{k_1r_1}{s_1} \\
&\quad + \frac{k_1^2r_1^2}{s_1^2} - \frac{k_1^2r_1r_2}{s_1^2} - \frac{k_1^2t_1r_2v_2}{s_1^2} - \frac{k_1r_2}{s_1} - \frac{k_1^2r_1r_2}{s_1^2} + \frac{k_1^2r_2^2}{s_1^2}] \\
&= \frac{s_1}{k_1}[\frac{k_1^2t_1^2v_1v_2 + k_1t_1s_1v_1 + k_1^2t_1r_1v_1 - k_1^2t_1r_2v_1 + k_1t_1s_1v_2}{s_1^2} \\
&\quad \frac{+s_1^2 + k_1r_1s_1 - k_1r_2s_1 + k_1^2t_1r_1v_2 + k_1r_1s_1 + k_1^2r_1^2 - k_1^2r_1r_2}{s_1^2} \\
&\quad \frac{-k_1^2t_1r_2v_2 - k_1r_2s_1 - k_1^2r_1r_2 + k_1^2r_2^2}{s_1^2}] \\
&= \frac{k_1^2t_1^2v_1v_2 + k_1t_1s_1v_1 + k_1^2t_1r_1v_1 - k_1^2t_1r_2v_1 + k_1t_1s_1v_2 + s_1^2}{s_1k_1} \\
&\quad \frac{+k_1r_1s_1 - k_1r_2s_1 + k_1^2t_1r_1v_2 + k_1r_1s_1 + k_1^2r_1^2 - k_1^2r_1r_2}{s_1k_1} \\
&\quad \frac{-k_1^2t_1r_2v_2 - k_1r_2s_1 - k_1^2r_1r_2 + k_1^2r_2^2}{s_1k_1} \\
&= \frac{k_1t_1^2v_1v_2}{s_1} + t_1v_1 + \frac{k_1t_1r_1v_1}{s_1} - \frac{k_1t_1r_2v_1}{s_1} + t_1v_2 \\
&\quad + \frac{s_1}{k_1} + r_1 - r_2 + \frac{k_1t_1r_1v_2}{s_1} + r_1 + \frac{k_1r_1^2}{s_1} - \frac{k_1r_1r_2}{s_1} - \frac{k_1t_1r_2v_2}{s_1} - r_2 - \frac{k_1r_1r_2}{s_1} + \frac{k_1r_2^2}{s_1} \\
&= \frac{k_1t_1^2v_1v_2}{s_1} + t_1v_1 + \frac{k_1t_1r_1v_1}{s_1} - \frac{k_1t_1r_2v_1}{s_1} + t_1v_2 + \frac{s_1}{k_1} + 2\,r_1 - 2r_2 + \frac{k_1t_1r_1v_2}{s_1} + \frac{k_1r_1^2}{s_1} - 2\frac{k_1r_1r_2}{s_1} \\
&\quad - \frac{k_1t_1r_2v_2}{s_1} + \frac{k_1r_2^2}{s_1} \\
\frac{s_1e_{1_1}e_{2_3}}{k_1kst} &= \frac{s_1}{k_1kst}(\frac{k_1t_1v_1 + s_1 + k_1r_1 - k_1r_2}{s_1})(kst) \\
&= \frac{1}{k_1}(k_1t_1v_1 + s_1 + k_1r_1 - k_1r_2) = t_1v_1 + \frac{s_1}{k_1} + r_1 - r_2 \\
\frac{s_2e_{1_1}e_{2_2}}{k_2} &= \frac{s_2}{k_2}(\frac{k_1t_1v_1 + s_1 + k_1r_1 - k_1r_2}{s_1})(\frac{s_2 + k_2r_2 - k_2r_1}{s_2}) \\
&= \frac{s_2}{k_2}[(\frac{k_1t_1v_1}{s_1} + 1 + \frac{k_1r_1}{s_1} - \frac{k_1r_2}{s_1})(1 + \frac{k_2r_2}{s_2} - \frac{k_2r_1}{s_2})] \\
&= \frac{s_2}{k_2}[\frac{k_1t_1v_1}{s_1} + \frac{k_1k_2t_1r_2v_1}{s_1s_2} - \frac{k_1k_2t_1r_1v_1}{s_1s_2} + 1 + \frac{k_2r_2}{s_2} - \frac{k_2r_1}{s_2} + \frac{k_1r_1}{s_1} + \frac{k_1k_2r_1r_2}{s_1s_2} - \frac{k_1k_2r_1^2}{s_1s_2} \\
&\quad - \frac{k_1r_2}{s_1} - \frac{k_1k_2r_2^2}{s_1s_2} + \frac{k_1k_2r_1r_2}{s_1s_2}] \\
&= \frac{k_1t_1s_2v_1}{s_1k_2} + \frac{k_1t_1r_2v_1}{s_1} - \frac{k_1t_1r_1v_1}{s_1} + \frac{s_2}{k_2} + r_2 - r_1 \\
&\quad + \frac{k_1r_1s_2}{s_1k_2} + 2\frac{k_1r_1r_2}{s_1} - \frac{k_1r_1^2}{s_1} - \frac{k_1r_2s_2}{s_1k_2} - \frac{k_1r_2^2}{s_1} \\
\frac{s_2e_{1_1}e_{2_3}}{k_2kst} &= \frac{s_2}{k_2kst}(\frac{k_1t_1v_1 + s_1 + k_1r_1 - k_1r_2}{s_1})(kst) \\
&= (\frac{k_1t_1s_2v_1 + s_1s_2 + k_1s_2r_1 - k_1s_2r_2}{s_1k_2}) \\
&= \frac{k_1t_1s_2v_1}{s_1k_2} + \frac{s_2}{k_2} + \frac{k_1s_2r_1}{s_1k_2} - \frac{k_1s_2r_2}{s_1k_2}
\end{aligned}
$$

B. ORDER PRESERVING FEATURE CORRECTNESS

In MLS, data ordering is preserved using the data encryption function associated with: (i) the key generation conditions (Subsection IV-A), and (ii) the ω-concept that, although generating random values, retains the data ordering across the cyphertexts. The ω-concept uses a simple mathematical rule to ensure a "gap" between cyphertexts so that adding random offsets (sampled from a particular range) will not cause any overlap, and hence guarantees data ordering. The value for ω can be determined using 10^p, and the random values r_i can then be sampled from range 0 to ω. Random values are generated each time the encryption function is called, a side-effect of this is that data equality is not preserved. This feature facilitates precluding Cyphertext Only Attacks (COAs) by generating different cyphertexts for the same plaintext value, even when the same list of keys is used (the probabilistic feature of the MLS scheme). If we consider the situation where $q = 1$ and $m = 3$, the encryptions of v_1 and v_2 are $E_1 = \{e_{1_1}, e_{1_2}, e_{1_3}\}$ and $E_2 = \{e_{2_1}, e_{2_2}, e_{2_3}\}$ where:

$$e_{1_1} = \frac{k_1 t_1 v_1 + s_1 + k_1(r_{1_1} - r_{1_2})}{s_1}$$
$$e_{2_1} = \frac{k_1 t_1 v_2 + s_1 + k_1(r_{2_1} - r_{2_2})}{s_1}$$

As already noted, the encryption function selects a different value for r_i every time the encryption function is invoked, thus $r_{1_i} \neq r_{2_i}$. If $v_1 > v_2$. Thus, applying the encryption function, and since the k_1, s_1 and t_1 values are all positive, the consistent values (private keys) will be:

$$k_1 t_1 v_1 + s_1 > k_1 t_1 v_2 + s_1$$

Adding random numbers to the above might change the data ordering. The ω-concept is used to create a "gap" between every consecutive plaintext value so as to allow the addition of a random number while preserving the data ordering. The value of ω is embedded in t_1, when $q = 1$, in other words $t_1 = (s_1 + k_1) \times \omega$. Recall the value of t_1 is multiplied with the v in MLS encryption function. The value of the random numbers r_{1_i} and r_{2_i} are sampled from 0 to ω. Therefore, the maximum value of $r_{1_1} - r_{1_2}$ is ω and also the maximum value of $r_{2_1} - r_{2_2}$ is ω (less than the gap multiplied with v in encryption function). Recall that r_{i_1} when $q = 1$ is greater than the value of other random values. Therefore, the encrypted values of v_1 and v_2 can be compared:

$$\begin{aligned} k_1 v_1 &> k_1 v_2 \\ k_1 v_1 \omega &\gg k_1 v_2 \omega \quad \textit{where } \omega = 10^p \\ k_1 v_1 \omega(s_1 + k_1) &\gg k_1 v_2 \omega(s_1 + k_1) \\ k_1 v_1 t_1 &\gg k_1 v_2 t_1 \\ k_1 v_1 t_1 + s_1 &\gg k_1 v_2 t_1 + s_1 \end{aligned} \tag{9}$$

In mathematics if c is an integer number greater than 1, then $c\times$ num $\gg$ $c+$num; the ω-concept uses this basic mathematical rule so that if a random value, sampled from the range 0 to ω, is added to the two operands in (9) the data order will still hold.

$$\begin{aligned} k_1 v_1 t_1 + s_1 + k_1(r_{1_1} - r_{1_2}) &> k_1 v_2 t_1 + s_1 + k_1(r_{2_1} - r_{2_2}) \\ \frac{k_1 v_1 t_1 + s_1 + k_1(r_{1_1} - r_{1_2})}{s_1} &> \frac{k_1 v_2 t_1 + s_1 + k_1(r_{2_1} - r_{2_2})}{s_1} \\ e_{1_1} &> e_{2_1} \end{aligned} \tag{10}$$

As argued in MLS the data order is preserved in e_q sub-cyphertext.

REFERENCES

[1] J. Barnes, "Using azure ML studio," in *Microsoft Azure Essentials Azure Machine Learning*, 1st ed. Redmond, WA, USA: Microsoft Press, 2015, ch. 3, sec. 4, pp. 44–93. [Online]. Available: https://tinyurl.com/2cpfvyy3

[2] V. Lakshmanan, "Using azure ML studio," in *Data Science on the Google Cloud Platform*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2022, ch. 9, sec. 2, pp. 317–322. [Online]. Available: https://tinyurl.com/4f54s22m

[3] A. Kaplunovich and Y. Yesha, "Cloud big data decision support system for machine learning on AWS: Analytics of analytics," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 3508–3516.

[4] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," in *Proc. AAAI Conf. Artif. Intell.*, 2017, pp. 4278–4284.

[5] C. Alippi, S. Disabato, and M. Roveri, "Moving convolutional neural networks to embedded systems: The AlexNet and VGG-16 case," in *Proc. 17th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Apr. 2018, pp. 212–223.

[6] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[7] G. Ateniese, L. V. Mancini, A. Spognardi, A. Villani, D. Vitali, and G. Felici, "Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers," *Int. J. Secur. Netw.*, vol. 10, no. 3, pp. 137–150, 2015, doi: 10.1504/IJSN.2015.071829.

[8] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 1322–1333.

[9] A. K. Das, "European Union's general data protection regulation, 2018: A brief overview," *Ann. Library Inf. Stud.*, vol. 65, no. 2, pp. 139–140, Jun. 2018.

[10] L. O. Gostin, "National health information privacy: Regulations under the health insurance portability and accountability act," *J. Amer. Med. Assoc.*, vol. 285, no. 23, pp. 3015–3021, Jun. 2001, doi: 10.1001/jama.285.23.3015.

[11] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2009.

[12] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim, and J.-S. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30039–30054, 2022, doi: 10.1109/ACCESS.2022.3159694.

[13] S. Alex, K. J. Dhanaraj, and P. P. Deepthi, "Private and energy-efficient decision tree-based disease detection for resource-constrained medical users in mobile healthcare network," *IEEE Access*, vol. 10, pp. 17098–17112, 2022, doi: 10.1109/ACCESS.2022.3149771.

[14] M. Barni, C. Orlandi, and A. Piva, "A privacy-preserving protocol for neural-network-based computation," in *Proc. 8th Workshop Multimedia Secur.*, Sep. 2006, pp. 146–151.

[15] H. Fang and Q. Qian, "Privacy preserving machine learning with homomorphic encryption and federated learning," *Future Internet*, vol. 13, pp. 94–114, Apr. 2021, doi: 10.3390/fi13040094.

[16] Y. Lee, S. Heo, S. Cheon, S. Jeong, C. Kim, E. Kim, D. Lee, and H. Kim, "HECATE: Performance-aware scale optimization for homomorphic encryption compiler," in *Proc. IEEE/ACM Int. Symp. Code Gener. Optim. (CGO)*, Apr. 2022, pp. 193–204.

[17] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep neural networks over encrypted data," 2017, *arXiv:1711.05189*.

[18] Q. Liu, X. Lu, F. Luo, S. Zhou, J. He, and K. Wang, ''SecureBP from homomorphic encryption,'' *Secur. Commun. Netw.*, vol. 2020, pp. 1–9, Jun. 2020, doi: 10.1155/2020/5328059.

[19] I. Mustafa, H. Mustafa, A. T. Azar, S. Aslam, S. M. Mohsin, M. B. Qureshi, and N. Ashraf, ''Noise free fully homomorphic encryption scheme over non-associative algebra,'' *IEEE Access*, vol. 8, pp. 136524–136536, 2020, doi: 10.1109/ACCESS.2020.3007717.

[20] D. Liu, ''Homomorphic encryption for database querying,'' U.S. Patent US10 027 486 B2, Dec. 27, 2013.

[21] Y. Wang and Q. M. Malluhi, ''Privacy preserving computation in cloud using noise-free fully homomorphic encryption (FHE) schemes,'' in *Proc. ESORICS*, Heraklion, Greece, Sep. 2016, pp. 301–323.

[22] O. Özerk, C. Elgezen, A. Mert, E. Öztürk, and E. Savaş, ''Efficient number theoretic transform implementation on GPU for homomorphic encryption,'' *J. Supercomput.*, vol. 78, pp. 2840–2872, Jul. 2021, doi: 10.1007/978-3-319-45744-4-15.

[23] J. Bos, K. Lauter, J. Loftus, and M. Naehrig, ''Improved security for a ring-based fully homomorphic encryption scheme,'' in *Proc. IMACC*, Oxford, U.K., 2013, pp. 45–64.

[24] Z. Brakerski, ''Fully homomorphic encryption without modulus switching from classical GapSVP,'' in *Proc. Crypto*, Santa Barbara, CA, USA, 2012, pp. 868–886, doi: 10.1007/978-3-642-32009-5-50.

[25] M. Dijk, C. Gentry, S. Halevi, and V. Vaikuntanathan, ''Fully homomorphic encryption over the integers,'' in *Proc. EUROCRYPT*, 2010, pp. 24–43, doi: 10.1007/978-3-642-13190-5-2.

[26] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, ''(Leveled) fully homomorphic encryption without bootstrapping,'' *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014, doi: 10.1145/2633600.

[27] A. Kipnis and E. Hibshoosh, ''Efficient methods for practical fully homomorphic symmetric-key encryption, randomization and verification,'' in *Proc. IACR*, Nov. 2012.

[28] D. Liu, ''Practical fully homomorphic encryption without noise reduction,'' in *Proc. IACR*, May 2015.

[29] L. J. Aslett, P. M. Esperança, and C. Holmes, ''Encrypted statistical machine learning: New privacy preserving methods,'' 2015, *arXiv:1508.06845*.

[30] C. Orlandi, A. Piva, and M. Barni, ''Oblivious neural network computing via homomorphic encryption,'' *EURASIP J. Inf. Secur.*, vol. 2007, pp. 1–11, Jun. 2007, doi: 10.1155/2007/37343.

[31] H. Kumarage, I. Khalil, A. Alabdulatif, Z. Tari, and X. Yi, ''Secure data analytics for cloud-integrated Internet of Things applications,'' *IEEE Cloud Comput.*, vol. 3, no. 2, pp. 46–56, Mar. 2016, doi: 10.1109/MCC.2016.30.

[32] *IEEE Standard for Floating-Point Arithmetic*, Standard 754–2019, Revision IEEE 754-2008, Jul. 2019, pp. 1–84, doi: 10.1109/IEEESTD.2019.8766229.

[33] S. Wagh, D. Gupta, and N. Chandran, ''SecureNN: 3-party secure computation for neural network training,'' in *Proc. PoPETs*, Stockholm, Sweden, 2019, pp. 26–49.

[34] T. Veugen, ''Improving the DGK comparison protocol,'' in *Proc. IEEE Int. Workshop Inf. Forensics Secur. (WIFS)*, Dec. 2012, pp. 49–54, doi: 10.1109/WIFS.2012.6412624.

[35] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, ''CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy,'' in *Proc. ICML*, New York, NY, USA, 2016, pp. 201–210.

[36] T. Graepel, K. Lauter, and M. Naehrig, ''ML confidential: Machine learning on encrypted data,'' in *Proc. ICISC*, Seoul South Korea, 2012, pp. 1–21.

[37] P. Mohassel and Y. Zhang, ''SecureML: A system for scalable privacy-preserving machine learning,'' in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 19–38.

[38] N. Agrawal, A. S. Shamsabadi, M. J. Kusner, and A. Gascón, ''QUOTIENT: Two-party secure neural network training and prediction,'' in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1231–1247.

[39] R. Livni, S. Shalev-Shwartz, and O. Shamir, ''On the computational efficiency of training neural networks,'' in *Proc. Adv. NIPS*, Cambridge, MA, USA, 2014, pp. 855–863.

[40] H. Chabanne, A. D. Wargny, J. Milgram, C. Morel, and E. Prouff, ''Privacy-preserving classification on deep neural network,'' Tech. Rep., 2017.

[41] E. Hesamifard, H. Takabi, M. Ghasemi, and N. W. Rebecca, ''Privacy-preserving machine learning as a service,'' *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 3, pp. 123–142, Jun. 2018.

[42] D. Liu, E. Bertino, and X. Yi, ''Privacy of outsourced k-means clustering,'' in *Proc. 9th ACM Symp. Inf., Comput. Commun. Secur.*, Jun. 2014, pp. 123–134.

[43] J. W. Bos, K. Lauter, and M. Naehrig, ''Private predictive analysis on encrypted medical data,'' *J. Biomed. Informat.*, vol. 50, pp. 234–243, Aug. 2014, doi: 10.1016/j.jbi.2014.04.003.

[44] J. Yuan and S. Yu, ''Privacy preserving back-propagation neural network learning made practical with cloud computing,'' *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 1, pp. 212–221, Jan. 2014, doi: 10.1109/TPDS.2013.18.

[45] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, ''Learning internal representations by error propagation,'' in *Readings in Cognitive Science*, 1st ed. Univ. California San Diego, CA, USA: MIT Press, 1986, ch. 8, pp. 318–362. [Online]. Available: https://ieeexplore.ieee.org/document/6302929

[46] D. Dua and C. Graff, *UCI Machine Learning Repository*. Irvine, CA, USA: University of California, School of Information and Computer Science, 2019. [Online]. Available: http://archive.ics.uci.edu/ml

[47] J. Waldo, ''Remote procedure calls and Java Remote Method Invocation,'' *IEEE Concurrency*, vol. 6, no. 3, pp. 5–7, Jul. 1998.

[48] J. Makhoul, F. Kubala, R. Schwartz, and R. Weischedel, ''Performance measures for information extraction,'' in *Proc. DARPA Broadcast News Workshop*, Herndon, VA, USA, Feb. 1999, pp. 249–252.

[49] B. Karlik and A. V. Olgac, ''Performance analysis of various activation functions in generalized MLP architectures of neural networks,'' *Int. J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, Feb. 2011.

NAWAL ALMUTAIRI has a general background in data mining, security, and AI. She works in the field of cybersecurity, especially detecting attacks using ML. She is currently an Assistant Professor with the Information Technology Department, King Saud University. Her research interests include privacy-preserving data mining (PPDM), homomorphic encryption (HE), property preserving encryption (PPE), data mining as a service (DMaaS) using cloud facilities, and collaborative data mining.

FRANS COENEN has a general background in AI. He has been working in the field of data mining and knowledge discovery in data (KDD) for the last 15 years. He is currently a Professor with the Department of Computer Science, University of Liverpool, where he is the Director of Doctoral Network in AI for Future Digital Health. He also leads a small research group working on many aspects of data mining and KDD. He has some 390 refereed publications on KDD and AI related research. He has been on the programme committees for many KDD conferences and related events. His research interests include the application of the techniques of data mining and knowledge discovery in data to unusual data sets, such as graphs and social networks, time series, free text of all kinds, 2D and 3D images, particularly medical images, video data, and data mining over encrypted data.

KEITH DURES is currently a Lecturer in computer science with the Department of Computer Science, University of Liverpool, and the Assistant Director of Studies of Online M.Sc. Programmes. He is also the Chair of the IT Sub-Group (university-wide), an Admissions Tutor, an Internal Examiner, a Module Coordinator, and the CPD Lead of the Department of Computer Science. He is a member of the Teaching and Scholarship in Computing (TASC) Group with interests in research and development with respect to teaching and learning. His research interests include knowledge discovery in databases, data mining, software engineering, and cyber security. His professional exposure (includes British Computer Society and Higher Education Academy), includes responsibility for academic and commercial course development, teaching, supervision, and examination at all levels.