ML Confidential: Machine Learning on Encrypted Data

Thore Graepel
Microsoft Research
Cambridge, UK
thoreg@microsoft.com

Kristin Lauter
Microsoft Research
Redmond, WA
klauter@microsoft.com

Michael Naehrig
Eindhoven University of Technology
Eindhoven, The Netherlands
michael@cryptojedi.org

Abstract

We demonstrate that by using a recently proposed somewhat homomorphic encryption (SHE) scheme it is possible to delegate the execution of a machine learning (ML) algorithm to a compute service while retaining confidentiality of the training and test data. Since the computational complexity of the SHE scheme depends primarily on the number of multiplications to be carried out on the encrypted data, we devise a new class of machine learning algorithms in which the algorithm's predictions viewed as functions of the input data can be expressed as polynomials of bounded degree. We propose confidential ML algorithms for binary classification based on polynomial approximations to least-squares solutions obtained by a small number of gradient descent steps. We present experimental validation of the confidential ML pipeline and discuss the trade-offs regarding computational complexity, prediction accuracy and cryptographic security.

1 Introduction

Cloud service providers leverage their large investments in data centers to offer services which help smaller companies cut their costs. But one of the barriers to adoption of cloud services is concern over the privacy and confidentiality of the data being handled by the cloud, and the commercial value of that data or the regulations protecting the handling of sensitive data. In this work we propose a cloud service which provides confidential handling of machine learning tasks for various applications. Machine learning consists of two stages, the training stage and the classification stage, either or both of which can be outsourced to the cloud. In addition, when both stages are outsourced to the cloud, we propose an intermediate probabilistic verification stage to test and validate the learned model which has been computed by the cloud service. In the protocols we describe here, we identify three parties: the Data Owner, the Cloud Service Provider, and the Content Providers.

The Data Owner is the customer for the Cloud Service, and owns or is responsible for the data being processed. Content Providers upload data to the cloud, data which belongs to or is intended for the Data Owner. Content Providers could be, for example, remote devices, sensors or monitors which belong to the Data Owner, and which may have been provisioned by the Data Owner, for example with secret keys if using a symmetric key system. A typical scenario might be a patient who is the Data Owner, and Content Providers which consist of multiple health monitoring devices provisioned to monitor the patient's health and upload data to the Cloud Service. Alternatively, the Data Owner could be some large company with many lab technicians, partners, or contracted Content Providers

which upload data to the Cloud Service on behalf of the company, for example in the financial, pharmaceutical, or social media industry. The Cloud Service may be run by a third party, a partner company, or even the company itself, off-premises or in some stand-alone facility.

Our rationale for proposing these protocols is that there are some scenarios where outsourcing *computation* to a Cloud Service makes sense from a practical and rational economic point of view. Namely, when data is collected or uploaded from many diverse sources or parties, an online service can host the collection, storage, and computation of and on this data without requiring interaction with the data owner. This service allows the data owner to access and query their potentially large amount of data at any time from a device with little computational or storage capacity. The Data Owner may subsequently designate privileges to other parties (such as a health care provider) to access the data or to receive alerts or updates concerning some other processed form of the data. When outsourcing computation to a service makes sense, *and* confidentiality of the data is an issue, then our protocols for providing confidential processing of sensitive data are relevant.

One way to preserve confidentiality of data when outsourcing computation is to encrypt the data before uploading it to the cloud. This may limit the utility of the data, but recent advances in cryptography allow searching on encrypted data and performing operations on encrypted data, all *without decrypting* it. An encryption scheme which allows arbitrary additions and multiplications of ciphertexts is called Fully Homomorphic Encryption (FHE). If we fix in advance a bound, p, on the degree of the polynomials which can be evaluated on the ciphertexts, such a scheme is called Somewhat Homomorphic (SHE). The first FHE scheme was constructed by Gentry [9], and subsequent schemes [19, 4, 3, 10] have rapidly become more practical, with improved performance and parameters. Current solutions for FHE all have an underlying SHE scheme as a building block, and use various techniques such as bootstrapping to extend it to an FHE scheme. Recent schemes are based on computational hardness assumptions for problems related to well known lattice problems such as the Shortest Vector Problem (SVP). Specifically, Ring Learning With Errors (RLWE)-based schemes operate in polynomial rings, where polynomials can alternatively be viewed as vectors in a lattice, and these polynomial rings are "truncated" in several ways to make them into finite objects. It was shown in [15] how the hardness of the RLWE problem is related to SVP.

In practice, it was observed in [13] that many useful services can be provided built on functions which can be evaluated using only SHE schemes with a small number of multiplications. In this paper, we design confidential machine learning algorithms based on low-degree polynomial versions of classification algorithms. Section 2 is devoted to explaining low-degree polynomial versions of the classification algorithms. Section 3 describes the Somewhat Homomorphic Encryption scheme which we use. Section 4 gives the ML Confidential protocol and discusses security. Section 5 describes our proof-of-concept implementation of the Division-Free Integer classification algorithms and gives initial performance numbers.

Connections between cryptography and machine learning have been considered for a long time (see, e.g., [18]), mostly with the view that they are inverses of one another in the sense that cryptography aims to prevent access to information whereas machine learning attempts to extract information from data. Note that the Confidential ML problem discussed in this paper is also loosely related to doing inference on differentially private data (see [20] and references therein), the difference being that in our case the Cloud Service performing the inference calculations is not even able to interpret the results of its analysis.

2 Polynomial Machine Learning

As discussed in the introduction, SHE schemes can be used to evaluate polynomials of limited degree p. For an ML algorithm this means that the predictions viewed as functions of the training and test data must be polynomials of limited degree p. Note that this restriction does not refer to the actual input-output mapping learned by the algorithm but to the dependency of the predictions on the training and test data. Straightforward implementation of certain machine learning algorithms require operations which are not currently possible on encrypted data, namely:

Comparison: Under SHE it is not permissible to perform a comparison $x > y$ for $x, y \in \mathbb{R}$. This rules out learning algorithms like the perceptron or the support vector machine because they derive their class labels from thresholding real numbers, the k-nearest neighbors classifier, which requires

ordering neighbors according to distance, and decision trees, which threshold features at the nodes of the tree.

Division: Under SHE it is not permissible to perform a division x/y for $x \in \mathbb{R}$ and $y \in \mathbb{R} \setminus \{0\}$. This rules out algorithms that rely on matrix inversion such as exact Fisher's linear discriminant for classification and the standard rule for determining the coefficients in regression.

Other non-polynomial functions: Under SHE it is not permissible to apply arbitrary non-linear functions. This rules out methods like exact logistic or probit regression and non-linear neural networks which rely on the evaluation of sigmoidal functions, in particular bounded sigmoid functions which are hard to approximate with polynomials.

Given these limitations, we are still able to design non-trivial machine learning algorithms that can be implemented under SHE.

Definition 1 (Polynomial learning/prediction algorithm) *Let $A : (\mathbb{R}^n \times \mathcal{Y})^m \times \mathbb{R}^n \to \mathcal{Y}$ be a learning/prediction algorithm that takes a training sample $(\mathbb{R}^n \times \mathcal{Y})^m$ of size m and a test input $\mathbf{x} \in \mathbb{R}^n$ and returns a prediction $y \in \mathcal{Y}$. If the function A is at most a p-degree polynomial in its arguments, then we call the learning/prediction algorithm p-polynomial.*

This definition can be applied directly to regression learning algorithms where $\mathcal{Y} = \mathbb{R}^{n'}$, and tells us that exact least-squares linear regression is not p-polynomial due to the required matrix inversion. Note that classification algorithms cannot be p-polynomial by definition because they have discrete outputs $y \in \mathcal{Y}$. However, in this case we can still use the above definition as guidance if we decompose a classification algorithm as $A = g \circ f$, with a mapping $f : (\mathbb{R}^n \times \mathcal{Y})^m \times \mathbb{R}^n \to \mathbb{R}^{n'}$ to a vector of real-valued scores, and a discretization operation $g : \mathbb{R}^{n'} \to \mathcal{Y}$. This decomposition is possible for a large class of algorithms including Linear Discriminant Analysis and Support Vector Machines, and allows us to carry out the function f under SHE by the Cloud Service, and to carry out function g by the Data Provider. In the following, we focus on the task of binary classification, but note that tasks like regression and dimensionality reduction could be cast in a similar framework.

2.1 Classification

Let us consider the case of binary classification with inputs in $\mathbb{R}^n$ and binary target outputs from $\mathcal{Y} = \{+1, -1\}$. We consider a linear classifier of the form $A(\mathbf{x}; \mathbf{w}, c) := \mathrm{sign}(f(\mathbf{x}; \mathbf{w}, c))$ with the score function $f(\mathbf{x}; \mathbf{w}, c) := \mathbf{w}^T\mathbf{x} - c$. According to the Confidential ML protocol, we know the sets of positive and negative training examples, and their cardinalities, and can hence carry out operations on these sets separately. This leads us to consider the simple Linear Means and Fisher's Linear Discriminant classifiers, both of which require only class-conditional statistics to be evaluated. Note, that the server only returns the argument of the sign function for each test example, and that the client takes the sign to obtain the class label, because SHE does not enable comparison.

2.1.1 Linear Means Classifier

The Linear Means (LM) classifier determines $\mathbf{w}$ and c such that $f(\mathbf{x}; \mathbf{w}, c) = 0$ defines a hyper-plane midway on and orthogonal to the line through the two class-conditional means. It can be derived as the Bayes optimal decision boundary in the case that the two class-conditional distributions have identical isotropic Gaussian distributions [5].

Let $I_y := \{i \in \{1, \ldots, m\} | y_i = y\}$ be the index set of training examples with label y and let $m_y := \|I_y\|$. Calculate the class-conditional mean vectors as, $\mathbf{m}_y := m_y^{-1}\mathbf{s}_y$ with $\mathbf{s}_y := \sum_{i \in I_y} \mathbf{x}_i$, from which we obtain the weight vector as the difference vector between the two class-conditional means $\mathbf{w}^* := \mathbf{m}_{+1} - \mathbf{m}_{-1}$. The value of the threshold c is calculated using the condition $\mathbf{w}^{*T}\mathbf{x}_0 - c = 0$ for the mid-point, $\mathbf{x}_0 := (\mathbf{m}_{+1} + \mathbf{m}_{-1})/2$, between the two class means, which gives for the threshold: $c^* = (\mathbf{m}_{+1} - \mathbf{m}_{-1})(\mathbf{m}_{+1} + \mathbf{m}_{-1})/2$. For a given test example $\mathbf{x}$ the score $f^*(\mathbf{x}; \mathbf{w}^*, c^*) := \mathbf{w}^{*T}\mathbf{x} - c^*$ is a quadratic function in the training data and a linear function in the test example and the LM classifier is hence p-polynomial. The server would return the score f for each test example to the client, who would then discover the sign and hence the label of the returned value upon decryption, evaluating g.

2.1.2 Fisher's Linear Discriminant Classifier

Now that we have convinced ourselves that machine learning under SHE is possible, let us move on to a more demanding example, Fisher's linear discriminant (FLD) classifier [7]. This algorithm is similar to the Linear Means classifier, but does take into account the class-conditional covariances. It aims at finding a project that maximizes the separation between classes as the ratio S between the variance σ^2_{inter} between classes and the variance σ^2_{intra} within classes,

$$S := \frac{\sigma^2_{\text{inter}}}{\sigma^2_{\text{intra}}} = \frac{\mathbf{w}^T\mathbf{D}\mathbf{w}}{\mathbf{w}^T\mathbf{C}\mathbf{w}} \tag{1}$$

with $\mathbf{D} := \mathbf{d}\mathbf{d}^T$ and $\mathbf{d} := \mathbf{m}_{+1} - \mathbf{m}_{-1}$ and $\mathbf{C} := \mathbf{C}_{+1} + \mathbf{C}_{-1}$. Here, $\mathbf{C}_y := \frac{1}{m_y}\sum_{i\in I_y}(\mathbf{x}_i - \mathbf{m}_y)(\mathbf{x}_i - \mathbf{m}_y)^T$ is the class-conditional covariance matrix of the data. Taking the gradient w.r.t. $\mathbf{w}$ and setting it to zero shows that $\mathbf{w}^*$ is the solution of a generalized eigenvalue problem $\mathbf{D}\mathbf{w} = \lambda\mathbf{C}\mathbf{w}$. Since $\mathbf{D} = \mathbf{d}\mathbf{d}^T$ has rank one, we can write $\mathbf{D}\mathbf{w} = a\mathbf{d}$ for some $a \in \mathbb{R}$ and hence $\mathbf{C}\mathbf{w}^* \propto \mathbf{d}$. Determining $\mathbf{w}^*$ requires solving a linear system of equations. Unfortunately, calculating the inverse $\mathbf{C}^{-1}$ exactly requires division, which is not permissable under SHE. We therefore aim at solving the linear system approximately using a least-squares approach so as to obtain a p-polynomial learning/prediction algorithm. The straight-forward cost function is $E(\mathbf{w}) := \frac{1}{2}||\mathbf{C}\mathbf{w} - \mathbf{d}||^2$, but instead of the standard Euclidean norm, we choose $||\mathbf{v}||^2 := \mathbf{v}^T\mathbf{C}^{-1}\mathbf{v}$ for better conditioning. Then the gradient is $\nabla_{\mathbf{w}}E(\mathbf{w}) = \mathbf{C}\mathbf{w} - \mathbf{d}$ and we can use gradient-based methods to find the solution $\mathbf{w}^*$. Once $\mathbf{w}$ has been found the threshold can be chosen as $c^* := \mathbf{w}^{*T}(\mathbf{m}_{+1} + \mathbf{m}_{-1})/2$.

The challenge then is to approximately solve a linear system using as few multiplications as possible. For the sake of illustration, let us consider standard gradient descent with a fixed learning rate η. If we define $\mathbf{R} := \mathbf{I} - \eta\mathbf{C}$ and $\mathbf{a} := \eta\mathbf{d}$ we obtain the well-known recursion $\mathbf{w}_{j+1} = \mathbf{R}\mathbf{w}_j + \mathbf{a}$ Defining $\mathbf{w}_0 = \mathbf{0}$ we can express the rth order approximation $\mathbf{w}_r$ of $\mathbf{w}^*$ as

$$\mathbf{w}_r := \left(\sum_{j=0}^{r-1}\mathbf{R}^j\right)\mathbf{a} = \eta\left(\sum_{j=0}^{r-1}(\mathbf{I} - \eta\mathbf{C})^j\right)\mathbf{d}\,. \tag{2}$$

This series converges if the spectral radius of $\mathbf{R}$ is less than one, i.e., if the absolute value of its largest eigenvalue is less than one, which can be ensured by choosing η sufficiently small. Depending on the order of approximation r, we obtain a p-polynomial FLD algorithm with $p = 2(r-1)+1$. Note that the sufficient statistics for the FLD algorithm are the class-conditional means $\mathbf{m}_y$ and covariance matrices $\mathbf{C}_y$. If it is desired to reduce the required communication overhead at the cost of increasing the Data Provider workload, then instead of transmitting the raw training data to the Cloud Provider, the Data Provider can calculate and transmit the sufficient statistics for the training data instead.

2.2 Other Machine Learning Tasks and Generalization Properties

While we focus on binary classification in this paper, it is certainly possible to extend our methodology to other machine learning tasks including regression, dimensionality reduction, and clustering. In particular, the case of multivariate linear regression is quite similar to FLD in that the exact solution requires a matrix inverse, which can be approximated using gradient descent. Also, principal component analysis (PCA) [11], which is probably the most popular method for dimensionality reduction, can be expressed as a least squares problem the solution of which can be approximated by gradient descent. Clustering may well be the most difficult task in this context, but it would appear that spectral clustering solutions [16] could be approximated in a similar way.

Another interesting aspect of polynomial machine learning are its generalization properties. Although in Confidential ML algorithms the hypothesis class (e.g., linear classifiers) remains the same with respect to the exact algorithm, the properties of SHE require us to produce predictions which are polynomials of limited degree in the input data. As a consequence, the set of hypotheses that can be reached by a p-polynomial learning algorithm is very limited. One would expect that this limited capacity would have a positive effect on the generalization ability. While we do not have any formal results on this, we believe it may be possible to formalize this idea based on the stability bounds on the generalization error in [17], because the approximations required by SHE can be viewed as a specific form of "early stopping".

3 Somewhat Homomorphic Encryption

In this section, we describe a somewhat homomorphic public-key encryption (SHE) scheme based on the Ring Learning With Errors (RLWE) problem [15]. It can be used to realize low degree confidential machine learning algorithms as described in Section 2. It extends the encryption scheme in [15] and resembles the SHE scheme from [2] in the RLWE case, as recently described in [6].

For simplicity and later reference in the description of our experiments, we discuss a special case of the scheme, for more details see [15, 2, 6]. Ciphertexts consist of polynomials in the ring $R = \mathbb{Z}[x]/(f(x))$, where $f(x) = x^d + 1$ and $d = 2^k$, i.e. integer polynomials of degree at most $d-1$. Note that f is the $2d$-th cyclotomic polynomial. Computations in R are done by the usual polynomial addition and multiplication with results reduced modulo $f(x)$. We fix an integer modulus $q > 1$ and denote by R_q the set of polynomials in R with coefficients in $(-q/2, q/2]$. For $z \in \mathbb{Z}$ denote by $[z]_q$ the unique integer in $(-q/2, q/2]$ with $[z]_q \equiv z \pmod{q}$. The message space is the set R_t for another integer modulus $t > 1$ ($t < q$). We use the same notation with q replaced by t. Thus, messages to be encrypted under the SHE scheme are polynomials of degree at most $d-1$ with integer coefficients in $(-t/2, t/2]$. Let $\Delta = \lfloor q/t \rfloor$ be the largest integer less than or equal to q/t. When applied to a polynomial $g \in R$, $\lfloor g \rfloor$ means rounding down coefficient-wise. We also use the notation $\lfloor \cdot \rceil$ for rounding to the nearest integer. As error distribution we take the discrete Gaussian distribution $\chi = D_{\mathbb{Z}^d,\sigma}$ with standard deviation σ over R. The parameters d, q, t and σ need to be chosen in a way to guarantee correctness, i.e. that decryption works correctly, and security. Section 5 below gives such concrete parameters. Given the above setting (following notation in [6]), we now describe the SHE scheme with algorithms for key generation, encryption, addition, multiplication, and decryption.

$\mathbf{SH.Keygen}$: The key generation algorithm samples $s \leftarrow \chi$ and sets the *secret key* $\mathsf{sk} := s$. It samples a uniformly random ring element $a_1 \leftarrow R_q$ and an error $e \leftarrow \chi$ and computes the *public key* $\mathsf{pk} := (a_0 = [-(a_1 s + e)]_q, a_1)$.

$\mathbf{SH.Enc}(\mathsf{pk}, m)$: Given the public key $\mathsf{pk} = (a_0, a_1)$ and a message $m \in R_t$, encryption samples $u \leftarrow \chi$, and $f, g \leftarrow \chi$, and computes the ciphertext $\mathsf{ct} = (c_0, c_1) := ([a_0 \cdot u + g + \Delta \cdot m]_q, [a_1 \cdot u + f]_q)$.

Note that a homomorphic multiplication increases the length of a ciphertext. Using relinearization techniques, it can be reduced to a two-element ciphertext again (see e.g. [13, 6]). For the purpose of this paper, we do not consider relinearization, thus ciphertexts can have more than two elements and we describe decryption and homomorphic operations for general ciphertexts.

$\mathbf{SH.Dec}(\mathsf{sk}, \mathsf{ct} = (c_0, c_1, \ldots, c_k))$: Decryption computes $[\lfloor t \cdot [c_0 + \mathsf{sk} \cdot c_1 + \ldots + \mathsf{sk}^k \cdot c_k]_q / q \rceil]_t$.

In general, the homomorphic operations SH.Add and SH.Mult get as input two ciphertexts $\mathsf{ct} = (c_0, c_1, \ldots, c_k)$ and $\mathsf{ct}' = (c'_0, c'_1, \ldots, c'_l)$, where w.l.o.g. $k \geq l$. The output of SH.Add contains $k+1$ ring elements, whereas the output of SH.Mult contains $k+l+1$ ring elements.

$\mathbf{SH.Add}(\mathsf{pk}, \mathsf{ct}_0, \mathsf{ct}_1)$: Let $\mathsf{ct}_1 = (c_0, c_1, \ldots, c_k)$ and $\mathsf{ct}_2 = (d_0, d_1, \ldots, d_l)$. Homomorphic addition is done by component-wise addition $\mathsf{ct}_{\mathsf{add}} = (c_0 + d_0, c_1 + d_1, \ldots, c_l + d_l, c_{l+1}, \ldots, c_k)$.

$\mathbf{SH.Mult}(\mathsf{pk}, \mathsf{ct}_0, \mathsf{ct}_1)$: Let $\mathsf{ct}_1 = (c_0, c_1, \ldots, c_k)$, $\mathsf{ct}_2 = (d_0, d_1, \ldots, d_l)$ and consider the polynomials $\mathsf{ct}_1(X) = c_0 + c_1 X + \ldots + c_k X^k$ and $\mathsf{ct}_2(X) = d_0 + d_1 X + \ldots + d_l X^l$ over R. The homomorphic multiplication algorithm computes the polynomial product

$$\mathsf{ct}_1(X) \cdot \mathsf{ct}_2(X) = e_0 + e_1 X + \ldots + e_{k+l+1} X^{k+l+1} \tag{3}$$

in the polynomial ring $R[X]$ over R. The output ciphertext is $\mathsf{ct}_{\mathsf{mlt}} = (\lfloor t \cdot e_0 / q \rceil, \ldots, \lfloor t \cdot e_{k+l+1} / q \rceil)$.

This SHE scheme has been recently described and analysed in [6] and is closely related to the scheme in [4] and [13]. We refer to these papers for correctness and security under the RLWE assumption. However note that the evaluation of the ciphertext polynomial at the secret key (as computed during decryption) can be written as $[\mathsf{ct}(\mathsf{sk})]_q = [\Delta \cdot m + v]_q$, where v is a noise term that grows during homomorphic operations. Only if v is small enough, the ciphertext still decrypts correctly. How quickly v grows with each multiplication and addition determines the capabilities of the SHE scheme. An advantage of the present scheme is that the factor by which v grows is independent of the input ciphertext noise (see [2, 6]).

Encoding real numbers. In order to do meaningful computations for ML, we would ideally like to do computations on real numbers, i.e. we need to encode real numbers as elements of R_t. Homomor-

phic operations under SHE encryption correspond to polynomial operations in R with coefficients modulo t. To reflect addition and multiplication of given numbers by the corresponding polynomial operations, we resort to the method in [13, Section 4] for encoding integers. We first represent a real number by an integer value. Since any real number can be approximated by rational numbers to arbitrary precision, we can fix a desired precision, multiply through by a fixed denominator, and round to the nearest integer.

An integer value z is encoded as an element $m_z \in R_t$ by using the bits in its binary representation as the coefficients of m_z. This means we use the following encoding function:

$$\text{encode} : \mathbb{Z} \to R_t,\ z = \text{sign}(z)(z_s, z_{s-1}, \ldots, z_1, z_0)_2 \mapsto m_z = \text{sign}(z)(z_0 + z_1 x + \ldots + z_s x^s).$$

To get back a number encoded in a polynomial, we evaluate it at $x = 2$. For the polynomial operations in R_t to reflect integer addition or multiplication, it is important that no reductions modulo t or modulo f occur. A multiplication after which a reduction modulo f is done does not correspond to integer multiplication of the encoded numbers any more. The same holds for reductions modulo t. The value t must therefore be large enough that all coefficients of polynomials representing values in the ML algorithm do not grow out of $(-t/2, t/2]$. Also the initial polynomial degree of encoded integers (i.e. their bit size) must be small enough so that the resulting polynomials after all multiplications still have degree less than d.

4 Protocol and Security considerations

Three types of parties interact in the protocol: Data Owner, Cloud Service Provider, and Content Providers. It can be implemented using either a private key or a public key SHE scheme.

ML Confidential Protocol

Key Generation (Private and Public Key versions). The Data Owner executes the $\mathsf{SH.Keygen}$ protocol for either a private key or a public key version of the SHE scheme. For the private key version, the Data Owner shares the private key with the Content Providers and stores it locally. For the public key version, the Data Owner publishes the public key and stores the private key locally.

Encryption (Content Providers encrypt confidential, labeled data to upload to the Cloud). For all training vectors in a class y, $\mathbf{x} \in I_y$, the Content Providers encrypt $\mathbf{x}$ and send $\mathsf{SH.Enc}(\mathsf{pk}, \mathbf{x})$ to the Cloud. Alternatively, the Content Providers may encrypt preprocessed versions of the training sets, e.g. synthetic data such as the sum or the covariance matrix of a class (i.e. the sufficient statistics).

Training (Cloud computes encrypted form of Learned Model). Training vectors consisting of encrypted, labeled content, $\mathsf{SH.Enc}(\mathsf{pk}, \mathbf{x})$, are processed by the Cloud using the algorithms in Section 5.1 to compute an encrypted form of the Learned Model. Section 5.1 gives Division-Free, Integer versions of the LM and FLD classifiers, which the Cloud can implement on the encrypted inputs using $\mathsf{SH.Add}$ and $\mathsf{SH.Mult}$. The Cloud stores and returns the encrypted Learned Model.

Verification of Learned Model (Probabilistic testing of the Learned Model). The Data Owner encrypts test vectors $\mathsf{SH.Enc}(\mathsf{pk}, \mathbf{x})$ for which it knows the classifications, and sends them to the Cloud. The Cloud performs the classification algorithm and returns encrypted classification results to the Data Owner. The Data Owner decrypts the results and compares with the known classification labels to assess the test error of the Learned Model in the Cloud.

Classification New test vectors (not used in the training stage) are sent to the Cloud by the Data Owner or the Content Providers and encrypted classifications are returned to the Data Owner. The Data Owner decrypts the results to obtain the classifications.

Security Considerations. The protocol assumes a model in which the Cloud is an *Honest but Curious* party, i.e. the Cloud will follow the stated protocol to provide the desired functionality, and will not deviate nor fail to provide the service or return results, but that it is *Curious* in the sense that it would look at available information. This assumption is reasonable to model a rational, economically motivated cloud service provider: the Cloud is motivated to provide excellent service, and yet would be motivated to take advantage of extra available information. A *Malicious* Cloud is a much stronger adversary, who would potentially mishandle calculations, delete data, refuse to return results, collude with other parties, etc. In most of these malicious behaviors, the Cloud would be likely to get caught, and thus damage its reputation if trying to run a successful business.

The verification step we propose is analogous to a naive version of Proof-of-Storage (PoS) protocols, since verification requires the Data Owner to store a certain number of labeled samples locally in order to be able to test correctness (and determine test error) of the Cloud's computations. Since we are assuming an Honest but Curious model for the Cloud, the Data Owner only needs to store a number of test examples that is sufficient statistically to determine the test error of the Cloud (or detect any accidental error). We are also implicitly assuming that the Content Providers do not behave maliciously, and correctly encrypt and upload data.

The Cloud must necessarily learn a certain amount of information in order to provide the functionality required, that is, computing a Learned Model from a collection of training vectors in Stage 1 and classifying test vectors in Stage 2. This includes knowing the number of vectors used in the training phase, and the number of test vectors submitted for classification. In addition, our scheme discloses the number of vectors within each class, and also the relative size of the entries in the test vectors can be deduced once the parameters for the SHE scheme and the number of test vectors are known.

The underlying SHE schemes are randomized and have semantic security against passive adversaries, a property which ensures that an adversary cannot distinguish an encryption of one message from another. The Cloud handles encrypted data and performs SHE operations, and using the public key, can encrypt messages of its choice. However, the Cloud does not obtain decryptions of the ciphertexts that it handles.

5 Proof of Concept and Experimental Results

In this section we provide experimental results at a small scale to show how confidential machine learning works in principle. Due to the rather high computational cost of SHE, we restrict ourselves to binary classification on two small data sets: an artificially generated data set and the well-known Iris data set first analyzed by Fisher in his seminal paper on linear discriminant analysis [7].

The surrogate data set was designed to illustrate how the approximate FLD classifier can improve upon the LM classifier. The data consists of 100 i.i.d. samples for each class $y \in \{-1, +1\}$ from the distributions $p(\mathbf{x}|y) = \mathcal{N}(\mathbf{x}; \mathbf{m}_y, \mathbf{C})$, where $\mathbf{m}_y := (0, 4y)^T$ and the covariance matrix $\mathbf{C}$ has elements $c_{11} = c_{22} = 8.5$ and $c_{12} = c_{21} = 7.5$, with eigenvalues $\lambda_1 = 16$ and $\lambda_2 = 1$ corresponding to eigenvectors $\mathbf{v}_1 = (1, 1)^T$ and $\mathbf{v}_2 = (1, -1)$, respectively. The Linear Means classifier on this data set has a very high expected error because it disregards the strong covariance structure in the data. In comparison, the FLD classifier is expected to perform much better.

The Iris data[1] consists of three classes of 50 examples each, each class referring to a type of iris plant. There are 4 real-valued input features, and we consider two classes, one corresponding to the class *Iris Setosa* and the other to the union of the classes *Iris Versicolour* and *Iris Virginica*.

5.1 Division-Free Integer ML Algorithms

Since we must transform real values into integers to encode them for SHE encryption, we must avoid divisions since there is no corresponding operation for encoded integers. Therefore, we describe division-free integer (DFI) versions of the LM classifier and FLD classifier described in Section 2.

For the LM Classifier we compute $m_{-1}\mathbf{s}_{+1}$ and $m_{+1}\mathbf{s}_{-1}$ and replace the weight vector by

$$\tilde{\mathbf{w}}^* := m_{-1}\mathbf{s}_{+1} - m_{+1}\mathbf{s}_{-1} = m_{+1}m_{-1}(\mathbf{m}_{+1} - \mathbf{m}_{-1}) = m_{+1}m_{-1}\mathbf{w}^*. \tag{4}$$

Similarly, the threshold is replaced by $\tilde{c}^* = 2m_{+1}^2 m_{-1}^2 c^*$ using $\tilde{\mathbf{x}}_0 := m_{-1}\mathbf{s}_{+1} + m_{+1}\mathbf{s}_{-1} = 2m_{+1}m_{-1}\mathbf{x}_0$. Given a test vector $\mathbf{x}$, we use the classifier $\tilde{f}^*(\mathbf{x}; \tilde{\mathbf{w}}^*, \tilde{c}^*) := 2m_{+1}m_{-1}\tilde{\mathbf{w}}^{*T}\mathbf{x} - \tilde{c}^*$, which simply computes a multiple of the LM classifier $f^*(\mathbf{x}; \mathbf{w}^*, c^*)$ with the same sign. The algorithm can be made confidential by encoding the real vector coefficients as integers. Then one encrypts the input vectors coefficient-wise and carries out the linear algebra operations with vectors of ciphertexts using SH.Add and SH.Mult.

A similar procedure is done for the FLD classifier. We use the same classifying function $\tilde{f}^*$ but with a different weight vector $\tilde{\mathbf{w}}^*$. We compute multiples of the class-conditional covariance matrices as

[1] Available at `http://archive.ics.uci.edu/ml/datasets/Iris` [8]

$\tilde{\mathbf{C}}_{+1} = m_{+1}^3\mathbf{C}_{+1}$ and $\tilde{\mathbf{C}}_{-1} = m_{-1}^3\mathbf{C}_{-1}$. In general, we compute $\tilde{\mathbf{C}} = m_{+1}^3\tilde{\mathbf{C}_{-1}} + m_{-1}^3\tilde{\mathbf{C}_{+1}} = m_{+1}^3 m_{-1}^3\mathbf{C}$. Whenever we can use equal size training classes, i.e. $m_{+1} = m_{-1}$, we can reduce the coefficients by a factor m_{+1}^3.

The gradient descent iteration is done with fixed step size η such that $\eta^{-1} \in \mathbb{Z}$. Taking good care of all denominators that need to be multiplied by, we can deduce that the division free integer gradient descent computes the r-th weight vector $\tilde{\mathbf{w}}_r$, which is $\tilde{\mathbf{w}}_r = (m_{+1}^3 m_{-1}^3 \eta^{-1})^r \mathbf{w}_r$, where $\mathbf{w}_r$ is the result of the r-th iteration described in Section 2.

5.2 Choice of SHE Parameters

In this subsection, we discuss the specific parameters chosen for our implementation. It has been recently shown in [12] that the hardness of the RLWE problem is independent of the form of the modulus. This means that security is not compromised by choosing q with a special structure. Using a power of 2 for q dramatically eases modular reduction when compared an implementation where q is prime. Therefore, as in [6] we choose both q and t as powers of 2, i.e. $\Delta = \lfloor q/t \rfloor = q/t$ is also a power of 2. We also use the optimization proposed in [6] to choose the secret key $\mathsf{sk} = s$ randomly with binary coefficients in $\{0, 1\}$.

To determine parameters that guarantee a certain level of security, one has to consider the best known algorithms to attack the scheme. Its security is assessed by the logarithm of the running time of such algorithms. A security level of ℓ bits means that the best known attacks take about 2^ℓ basic operations. We chose parameters considered secure under the distinguishing attack in [14], using the method described in [13, Section 5.1] and [6, Section 6]. For the exact details of the security evaluation, we refer to [14, 13, 6]. Security depends on the size of q, σ, and d, and for a given pair q, σ one can determine a lower bound for d.

Additional conditions follow from ensuring correctness of decryption. As long as the inherent noise in ciphertexts is bounded by $\Delta/2 = q/2t$, decryption works correctly. Since homomorphic computations increase the noise level, this bounds the number of computations from above. In the division free integer algorithms the encrypted numbers tend to grow with the number of operations due to multiplications by denominators. To ensure meaningful results, t needs to be greater than all the coefficients of message polynomials that are held and operated on in encrypted form. The size of the standard deviation for the error terms and the desired number of homomorphic operations bound Δ and therefore q from below. For our implementation, we determined these quantities experimentally and then chose the degree d according to the security requirements.

5.3 Implementation and Performance

We implemented the above described SHE scheme and the division-free integer ML algorithms under SHE in the computer algebra package Magma [1], using internal functions for polynomial arithmetic and modular reductions. Table 1 summarizes timings for the SHE operations. The confidential version of the DFI-LM classifier uses the first parameter set (P_1) for both data sets. Parameters (P_2) were chosen for the 1-step and (P_3) for the 2-step FLD method. Due to its higher complexity and the higher value for t it requires a much larger value for q.

		SH.Keygen	SH.Enc	SH.Dec(2)	SH.Dec(3)	SH.Add	SH.Mult
(P_1)	$q = 2^{128}, t = 2^{15}$ $\sigma = 16, d = 4096$	156	379	29	52	1	106
(P_2)	$q = 2^{252}, t = 2^{35}$ $\sigma = 8, d = 8192$	382	853	98	193	4	370
(P_3)	$q = 2^{340}, t = 2^{40}$ $\sigma = 8, d = 8192$	403	879	118	231	4	446

Table 1: Timing in ms for SHE operations key generation, encryption, decryption of 2- or 3-element ciphertexts, homomorphic addition and multiplication in Magma on a single core of an Intel Core i5 CPU650 @ 3.2 GHz. Parameters (P_1) and (P_2) have 128 bits of security with distinguishing advantage 2^{-64}. Security for (P_3) is around 80 bits due to small σ compared to q.

Table 2 gives timings for the cloud-side computations under SHE. For both data sets, we preprocessed the data by shifting the mean to 0 and scaling by the standard deviation. The transformation of real values to integers was done by multiplying by 100 (by 10 for FLD) and then rounding to the nearest integer. The confidential versions of the algorithms produce the exact same classification output as our non-encrypted reference implementation. Comparing the costs for the FLD classifier using 1-step and 2-step gradient descent illustrates the cost of obtaining a more precise classification through a better approximation of the exact weight vector. More iterations require more multiplications and larger parameters which in turn increase the cost for polynomial operations.

data	# features	algorithm	parameters	train	classify
surrogate	2	linear means	(P_1)	230	235
Iris	4	linear means	(P_1)	510	496
surrogate	2	1-step linear discriminant	(P_2)	58710	1490
surrogate	2	2-step linear discriminant	(P_3)	74770	2680

Table 2: Timing in ms for confidential DFI ML under SHE in Magma on a single core of an Intel Core i5 CPU650 @ 3.2 GHz, not measuring encryption, communication, decryption. Entries in the column "train" are the time taken for the training phase, i.e. to compute the classifier from the encrypted training data. Entries in the column "classify" are the times for classifying a test vector.

Homomorphic operations on encrypted data are computationally expensive, mainly due to the enormous size of polynomials dictated by security and correctness requirements. A single polynomial in R_q has d coefficients of size $\log(q)$, so for parameters (P_1), such an element is of size $4096 \cdot 128 = 2^{19}$ bits. Such sizes are mainly induced by the polynomial degree of the algorithm to be computed under SHE. A trade-off between un-encrypted computations by the Content Provider and encrypted computations in the cloud can lower the degree of functions, the size of polynomial coefficients and encrypted numbers. Assume that the Content Provider computes certain sufficient statistics such as the sum of data vectors and the class conditional covariance matrices for the FLD classifier and encrypts those instead of single data vectors. This not only removes these computations from the confidential algorithms lowering their polynomial degree, but also provides fresh encryptions of these data with much smaller inherent noise terms. As a result, SHE parameters can be made smaller, or the confidential part of the algorithm can be made more complex, e.g. allowing more iterations in the gradient descent.

6 Conclusions and Future Work

With advances in machine learning and cloud computing the enormous value of data for commerce, society, and people's personal lives is becoming more and more evident. In order to realize this value it will be crucial to make data available for analysis while at the same time protect it from unwanted access. In this paper, we pointed out a way to reconcile these two conflicting goals: Confidential Machine Learning. We formalize the problem in terms of a multi-party data machine learning scenario involving a Data Owner, Data Providers, and a Cloud Service Provider and describe the desired functionality and security properties. We showed that it is possible to implement Confidential ML based on a recently proposed Somewhat Homomorphic Encryption scheme, using polynomial approximations to known ML algorithms.

Homomorphic encryption is a rapidly advancing field and so we expect that more complex ML algorithms applied to larger data sets requiring fewer computational resources may soon be possible. For example, it should soon be possible to use kernel methods to derive low-degree polynomial machine learning algorithms implementing non-linear mappings. Other open problems include the question, which protocols will be useful in practical data analysis scenarios, and how the computational burden can be optimally distributed between cloud and client taking into account the cost of communication. Furthermore, one can imagine even more complex multi-party scenarios in which multiple data-owners (e.g., Amazon, Netflix, Google, Facebook) would like to provide inputs for a single machine learning problem (e.g., product recommendation) without disclosing their data.

References

[1] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *J. Symbolic Comput.*, 24(3-4):235–265, 1997. Computational algebra and number theory (London, 1993).

[2] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. Cryptology ePrint Archive, Report 2012/078, 2012. `http://eprint.iacr.org/`.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. Fully homomorphic encryption without bootstrapping. Cryptology ePrint Archive, Report 2011/277, 2011. `http://eprint.iacr.org/`.

[4] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 505–524. Springer, 2011.

[5] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. John Wiley and Sons, 2nd edition, 2000.

[6] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012. `http://eprint.iacr.org/`.

[7] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annual Eugenics*, 7(2):179–188, 1936.

[8] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[9] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[10] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. Cryptology ePrint Archive, Report 2011/680, 2011. `http://eprint.iacr.org/`.

[11] I.T. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[12] Adeline Langlois and Damien Stehlé. Hardness of decision (r)lwe for any modulus. Cryptology ePrint Archive, Report 2012/091, 2012. `http://eprint.iacr.org/`.

[13] Kristin Lauter, Michael Naehrig, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Cloud Computing Security Workshop*, CCSW '11, pages 113–124, New York, NY, USA, 2011. ACM.

[14] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for lwe-based encryption. In Aggelos Kiayias, editor, *CT-RSA*, volume 6558 of *Lecture Notes in Computer Science*, pages 319–339. Springer, 2011.

[15] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2010. Full version available at `http://eprint.iacr.org/2012/230`.

[16] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press, 2002.

[17] Tomaso Poggio, Ryan Rifkin, Sayan Mukherjee, and Partha Niyogi. General conditions for predictivity in learning theory. *Nature*, 428:419–422, 2004.

[18] Ronald L. Rivest. Cryptography and machine learning. In *Advances in Cryptology - ASIACRYPT 91*, pages 427–439. Springer, 1993.

[19] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[20] Oliver Williams and Frank McSherry. Probabilistic inference and differential privacy. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2451–2459. 2010.