Privacy-Preserving Machine Learning Applications

Ariel Akemi Todoki

A thesis
submitted in partial fulfillment of the
requirements for the degree of

Master of Science in Computer Science and Systems

University of Washington

2019

Committee:

Anderson Nascimento

Martine De Cock

Program Authorized to Offer Degree:
Computer Science and Systems

University of Washington

Abstract

Privacy-Preserving Machine Learning Applications

Ariel Akemi Todoki

Chair of the Supervisory Committee:
Dr. Anderson Nascimento
School of Engineering and Technology

Machine learning has its many applications in different areas of interest that involves huge amounts of data in order to learn how to recognize, predict, and classify. One such area is in text classification where a private text message (i.e. SMS message, tweet, email) needs to be scored against a text classification model that contains proprietary information. Another such area is in medical diagnosis involving patient data in the form of medical records, test results, images, and genome data. While machine learning can be very successful in predicting and classifying, it relies on learning from user data that is private, confidential, and could include personally identifiable information. In this thesis, we use privacy-preserving techniques to: (i) train a logistic regression model on breast cancer tissue data and (ii) classify private texts using an adaboost model or a logistic regression model such that patient/user data is kept private. These techniques are information-theoretically secure, meaning the security does not rely on computational complexities (which could be broken in the future), and allows us to benefit from machine learning on data that would otherwise be kept private or unshared. In general, privacy-preserving techniques for computations run slower than when computed in the clear. To the best of our knowledge, our implementations show excellent runtimes compared to previous works for their respective tasks with no loss of accuracy.

TABLE OF CONTENTS

Page

Chapter 1: Introduction 1

Chapter 2: High Performance Logistic Regression for Privacy-Preserving Genome Analysis 3

Chapter 3: Privacy-Preserving Classification of Personal Text Messages with Secure Multi-Party Computation 22

Chapter 4: Conclusion 41

4.1 Individual Contributions 41

4.2 Future Work 41

Chapter 5: Appendix A 43

ACKNOWLEDGMENTS

I wish to express sincere appreciation to Dr. Anderson Nascimento and Dr. Martine De Cock for all of their guidance and support, and to whom I credit with my success.

Chapter 1

INTRODUCTION

The prevalence and success of machine learning is shown through its wide usage in many different real world applications in all different industries such as healthcare, business, agriculture, transportation, and many others. Its success stems from the idea that machine learning is about recognizing patterns from training data, and then being able to predict the outcome of future data. While companies and individuals can benefit greatly from the use of machine learning, there can be concerns of privacy due to the type of information that is usually held in the data involved. Sometimes, these privacy concerns are from (i) individuals who do not want to release their personal information, (ii) legal restriction that don't allow release of certain information such as patient data from health care providers, or (iii) company's own proprietary information. Fortunately, there are techniques that can be used to allow training and scoring on data that is private such that none of the private data is learned from any party involved. One such technique is called secure multiparty computation.

Secure multiparty computation (SMC) is a field in cryptography that allows evaluation of a function where its inputs are secret values from different parties, such that the parties do not learn what the other party's secret values are. We work in a semi-honest (honest-but-curious) setting which defines that our protocols are secure and accurate from adversarial parties that follow the protocol, and any information that is learned through the computation process does not leak any information about the secret data values. Our SMC protocols work on secret shares of data, meaning that our protocols are information theoretically secure. This means that its security is not based on the hardness of a computational problem (which could be broken with a large amount of computational power). We can build privacy-preserving

machine learning training and scoring algorithms by breaking them down into computations that we can do securely. In this way, we can build secure protocols by using others as building blocks, with the simplest building blocks being addition and multiplication.

In this paper, we introduce two applications for privacy-preserving machine learning. The first application is training on gene data that is private to each data owner. We use SMC to train a logistic regression model on the private gene data. We improve on previous secure implementations with a new secure protocol for evaluating the activation function in a more efficient way that does not require a secure comparison protocol. The second application is classifying a private text message, with the example use case of classifying a tweet as hate speech or not. We use SMC to classify private text messages with a private classifier. It is important to note that this work deals with a private text that is unstructured, and therefore we also describe how to do private feature extraction. For both applications, our secure protocols result in no loss of accuracy as compared to an in-the-clear version. Therefore, we show that we can still benefit from training/scoring machine learning models on private data without revealing it.

RESEARCH

High Performance Logistic Regression for Privacy-Preserving Genome Analysis

Martine De Cock[1*†], Rafael Dowsley[2], Anderson Nascimento[1], Davis Railsback[1], Jianwei Shen[1] and Ariel Todoki[1]

[*]Correspondence:
mdecock@uw.edu
[1]School of Engineering and Technology, University of Washington Tacoma, 98402 Tacoma, WA, USA
Full list of author information is available at the end of the article
[†]Guest Professor at Ghent University

Abstract

Background: In biomedical applications, valuable data is often split between parties who cannot openly share the data because of privacy regulations and concerns. Training Machine Learning models on the joint data without violating privacy is a major technology challenge that can be addressed by combining techniques from Machine Learning and cryptography. When collaboratively training Machine Learning models with Secure Multiparty Computation, the price paid for keeping the data of the parties private is an increase in computational cost and runtime. A careful choice of Machine Learning techniques, algorithmic and implementation optimizations are a necessity to enable practical secure Machine Learning over distributed datasets. Such optimizations can be tailored to the kind of data and Machine Learning problem at hand.

Methods: Our setup involves Secure Two-Party Computation protocols, along with a trusted initializer to distribute correlated randomness to the two computing parties. We use a gradient descent based algorithm for training our logistic regression model, and we break down the algorithm into corresponding secure protocols. Our main contribution is a new protocol for computing the activation function that requires neither secure comparison protocols nor Yao's garbled circuits.

Results: For our largest dataset, we train a model that requires over 7 billion secure multiplications; the training completes in about 2.71 seconds in a local area network. This implementation won first place in Track 4 of the iDASH2019 secure genome analysis competition.

Conclusions: In this paper, we present a secure logistic regression training protocol and its implementation, with a new protocol to securely compute the activation function. To the best of our knowledge, we present the fastest existing Secure Multiparty Computation implementation for training logistic regression models on high dimensional data.

Keywords: Logistic regression; Gradient descent; Machine Learning; Secure Multiparty Computation

Background

Introduction

Machine Learning (ML) has many applications in the biomedical domain, such as diagnosis and personalized medicine. Biomedical datasets are typically characterized by high dimensionality, i.e. a high number of features such as lab test results or gene expression values, and low sample size, i.e. a small number of training examples corresponding to e.g. patients or tissue samples. Adding to these challenges, valu-

able training data is often split between parties (*data owners*) who cannot openly share the data because of privacy regulations and concerns. Due to these concerns, privacy-preserving solutions, using techniques such as secure Multi-Party Computation (MPC), become important so that this data can still be used to train ML models, classify a diagnosis, and in some cases even derive genomic diagnoses [1].

We tackle the problem of training a binary classifier on high dimensional gene expression data held by different data owners, while keeping the training data private. This work is directly inspired by Track 4 of the iDASH2019 secure genome analysis competition[1]. The iDASH competition is a yearly international competition for participants to create and implement privacy-preserving protocols for applications with genomic data. The goal is in evaluating the best known secure methods to solve real-world problems in handling genomic data. There are a total of four different tracks, where Track 4 invites participants to design MPC solutions for collaborative training of ML models by two or more data owners. One of the competition datasets consists of 470 training examples (records) with 17,814 numeric features, while the other consists of 225 training examples with 12,634 numeric features. An initial 5-fold cross-validation analysis in the clear, i.e. without any encryption, indicated that in both cases logistic regression (LR) models are capable of yielding the level of prediction accuracy expected in the competition, prompting us to investigate MPC based protocols for secure LR training.

The competition requirements imply the existence of multiple data owners who each send their training example(s) in an encrypted or secret shared form to *data processors* (computing nodes), as illustrated in Figure 4. The *honest-but-curious* data processors are not to learn anything about the data as they engage in computations and communications with each other. At the end, they disclose the trained classifier – in our case, the coefficients of the LR model – to the data owners. Since the data processors cannot learn anything about the values in the dataset, this implies that our protocol is applicable in a wide range of scenarios, independently of how the original data is split by ownership. Our protocol works in scenarios where the data is horizontally partitioned, i.e. when each data owner has different records of the data, such as data belonging to different patients. It also works in scenarios where the data is vertically partitioned, i.e. when each data owner has different field of the data, such as the expression values for different genes.

The main novelty points of our solution for private LR training over a distributed dataset are: (i) a new protocol for securely computing the activation function that avoids a full-fledged secure comparison protocols; (ii) a novel way of bundling bit decomposition instantiations; and (iii) several cryptographic engineering enhancements that together with the novel protocol for the activation function gave us the fastest privacy-preserving LR implementation in the world when run in local area networks. In summary, we designed a concrete solution for fast secure training of a binary classifier over gene expression data that meets the strict security requirements of the iDASH2019 competition. For our largest dataset, we train a model that requires over 7 billion secure multiplications and the training completes in about 2.71 seconds in a LAN.

[1]http://www.humangenomeprivacy.org/2019/competition-tasks.html, acc. on Nov 22, 2019

We first discuss our work as compared to others in our **Related Work** section. In the **Methods** section, we present preliminary information on MPC, describe the secure protocols that form the building blocks for our secure LR protocol, and finally describe the protocol itself. In the **Results** section we describe details of our implementation and runtime results for the overall protocol and microbenchmarks for our secure activation function protocol. In the **Discussion** section, we note possible future work to improve and extend our results, and finally in the **Conclusions** section we present our summary remarks.

Related Work

A variety of efforts have previously been made to train LR classifiers in a privacy-preserving (PP) way.

One scenario that was considered in previous works [2, 3, 4] is the setting in which a data owner holds the data while another party (the data processor), such as a cloud service, is responsible for the model training. These solutions usually rely on homomorphic encryption, with the data owner encrypting and sending their data to the data processor who performs computations on the encrypted data without having to decrypt it.

When the data is held by multiple data owners, they can either execute a MPC protocol among themselves to train the model, or delegate the computation to a set of data processors that run a MPC protocol.

Existing multiparty computation approaches to secure LR differ in the numerical optimization algorithms used for LR training and in the cryptographic primitives leveraged [5, 6, 7, 8]. The SPARK protocol [5] uses additive homomorphic encryption (Paillier cryptosystem) and uses Newton-Raphson as the numerical optimization algorithm to find the values of the weights that maximize the log-likelihood. The SPARK protocol can use the actual logistic function without approximating it at the cost of the plaintext data being horizontally partitioned and seen by the data processors. The two protocols from [7] rely on the Newton-Raphson method, both approximate the logistic function, and both use additive secret sharing. The first protocol includes the use of Yao's garbled circuits to compute the approximation of the logistic function, while the second protocol uses a Taylor approximation and Euler's method. The PrivLogit method [8] uses Yao's garbled circuits and Paillier encryption in their protocol which uses the Newton-Raphson method and a constant Hessian approximation to speed up computation. However, this protocol relies on the plaintext data being horizontally partitioned and seen by the data processors, which, like the work in [5], would not align with the iDASH2019 competition requirements. We also point out a protocol secure against active adversaries from SecureNN [9] for computing a ReLU. While we compute a different function (clipped ReLU), we share a similar idea that using the most significant bit of an input can tell us the output of the function.

The work closest to ours is SecureML [6], which was the fastest protocol for privately training LR models based on secure multiparty computations prior to this work. SecureML separates the data owners from the data processors, and uses mini-batch gradient descent. The main novelty points of the SecureML paper are a clipped ReLu activation function (see Figure 5(b)), a novel truncation protocol, and

a combination of garbled circuits and secret sharing based MPC in order to obtain a good trade-off between communication, computation and round complexities. The SecureML protocol is evaluated on a dataset with up to 5,000 features, while – to the best of our knowledge – the existing runtime evaluation of all other approaches for MPC based LR training is limited to 400 features or less [5, 7, 8]. The SecureML protocol is split into an offline and online phase (the offline phase runs first and generates multiplication triples).

Methods

Security Model

The security model in which we analyze our protocol is the Universal Composability (UC) framework [10] as it provides the strongest security and composability guarantees and is the gold standard for analyzing cryptographic protocols nowadays. Here we will only give a short overview of the UC framework (for the specific case of two-party protocols), and refer interested readers to the book of Cramer et al. [11] for a detailed explanation.

The main advantage of the UC framework is that the UC composition theorem guarantees that any protocol proven UC-secure can also be securely composed with other copies of itself and of other protocols (even with arbitrarily concurrent executions) while preserving its security. Such guarantee is very useful since it allows the modular design of complex protocols, and is a necessary condition for protocols executing in complex environments such as the Internet.

The UC framework first considers a real world scenario in which the two protocol participants (henceforth denoted Alice and Bob) interact between themselves and with an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$ (that captures all activity external to the current execution of the protocol). The environment $\mathcal{Z}$ gives the inputs and gets the outputs from Alice and Bob. The adversary $\mathcal{A}$ delivers the messages exchanged between Alice and Bob (thus modeling an adversarial network scheduling) and can corrupt one of the participants, in which case he gains the control over it. In order to define security, an ideal world is also considered. In this ideal world, an idealized version $\mathcal{F}$ of the functionality that the protocol is supposed to perform is defined. The ideal functionality receives the inputs directly from Alice and Bob, performs the computations locally following the primitive specification and delivers the outputs directly to Alice and Bob. A protocol π is said to UC-realize functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between: (1) an execution of the protocol π with participants Alice and Bob, and adversary $\mathcal{A}$; (2) and an ideal execution with dummy parties that only forward inputs/outputs, $\mathcal{F}$ and $\mathcal{S}$.

This work considers honest-but-curious, static adversaries (like most other privacy-preserving machine learning protocols in the literature), i.e., the adversary chooses the party that he wants to corrupt before the protocol execution and he also follows the protocol instructions (but try to learn additional information). We consider the trusted initializer model, in which a trusted initializer functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ (described in Figure 1) pre-distributes correlated randomness to Alice and Bob. A trusted initializer (TI) has been often used to enable highly efficient solutions both in the context of PPML [12, 13, 14, 15, 16] as well as in other applications,

Functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$

$\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ is parametrized by an algorithm $\mathcal{D}$ for sampling the correlated randomness. Upon initialization run $(D_A, D_B) \xleftarrow{\$} \mathcal{D}$, and deliver D_A to Alice and D_B to Bob.

Figure 1 The Trusted Initializer functionality.

e.g., [17, 18, 19, 20, 21, 22]. We should remark that the trusted initializer is not involved in any part of the computation other than the pre-distribution of correlated randomness, and does not learn any data from Alice or Bob.

Simplifications: In our proofs the simulation strategy is simple and will be described briefly: all the messages look uniformly random from the recipient's point of view, except for the messages that open some secret share to a party, but these ones can be easily simulated using the output of the respective functionalities. Therefore a simulator $\mathcal{S}$, having the leverage of being able to simulate the trusted initializer functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ in the ideal world, can easily perform a perfect simulation of a real protocol execution; therefore making the real and ideal worlds indistinguishable for any environment $\mathcal{Z}$. In the ideal functionalities the messages are public delayed outputs, meaning that the simulator is first asked whether they should be delivered or not (this is due to the modeling that the adversary controls the network scheduling). This fact as well as the session identifications are omitted from our functionalities' descriptions for the sake of readability.

Secret Sharing Based Secure Multiparty Computation

Our MPC solution is based on additive secret sharing over a ring $\mathbb{Z}_q$. When secret sharing a value $x \in \mathbb{Z}_q$, Alice and Bob receive shares x_A and x_B (respectively) that are chosen uniformly at random with the constraint that $x_A + x_B = x \mod q$. We denote the pair of shares by $[\![x]\!]_q$. All computations are modulo q and the modular notation is henceforth omitted for conciseness. Note that no information of the secret value x is revealed to either party holding only one share. This value can be revealed/opened to each party by combining both shares. Some operations on secret shared values can be computed locally with no communication. Let $[\![x]\!]_q$, $[\![y]\!]_q$ be secret shared values and c be a constant. Alice and Bob can perform the following operations locally:

- Addition ($z = x + y$): Each party locally adds its local shares of x and y in order to obtain a share of z. This will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + [\![y]\!]_q$.
- Subtraction ($z = x - y$): Each party locally subtracts its local shares of y from that of x in order to obtain a share of z. This will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q - [\![y]\!]_q$.
- Multiplication by a constant ($z = cx$): Each party multiplies its local share of x by c to obtain a share of z. This will be denoted by $[\![z]\!]_q \leftarrow c[\![x]\!]_q$
- Addition of a constant ($z = x + c$): Alice adds c to her share x_A of x to obtain z_A, while Bob sets $z_B = x_B$. This will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + c$.

Functionality $\mathcal{F}_{\mathsf{DMM}}$

$\mathcal{F}_{\mathsf{DMM}}$ runs with Alice and Bob and is parametrized by the size q of the ring and the dimensions (i, j) and (j, k) of the matrices.

Input: Upon receiving a message from Alice/Bob with its shares of $[\![X]\!]_q$ and $[\![Y]\!]_q$, verify if the share of X is in $\mathbb{Z}_q^{i \times j}$ and the share of Y is in $\mathbb{Z}_q^{j \times k}$. If it is not, abort. Otherwise, record the shares, ignore any subsequent message from that party and inform the other party about the receipt.

Output: Upon receipt of the shares from both parties, reconstruct X and Y from the shares, compute $Z = XY$ and create a secret sharing $[\![Z]\!]_q$ to distribute to Alice and Bob: a corrupt party fix its share of the output to any chosen matrix and the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

Figure 2 The distributed matrix multiplication functionality.

The secure multiplication of secret shared values (i.e., $z = xy$) cannot be done on local shares only, and it involves communication between the parties. To obtain an efficient secure multiplication solution, we use the multiplication triples technique that was originally proposed by Beaver [23]. We use a trusted initializer to pre-distribute the multiplication triples (which are a form of correlated randomness) to Alice and Bob. We use the same protocol π_{DMM} for secure (matrix) multiplication of secret shared values as in [15, 24] and denote by π_{DM} the protocol for the special case of multiplication of single elements and π_{IP} for the inner product. As shown in [15] the protocol π_{DMM} UC-realizes the distributed matrix multiplication functionality $\mathcal{F}_{\mathsf{DMM}}$ (described in Figure 2) in the trusted initializer model. The protocol is described here for completeness:

Protocol 1: Secure Distributed Matrix Multiplication Protocol π_{DMM}

Input : $[\![X]\!]_q$, $[\![Y]\!]_q$

Output: $[\![Z]\!]_q$ such that $Z = XY$

1. The protocol is parametrized by the size q of the ring and the dimensions (i, j) and (j, k) of the matrices. The trusted initializer chooses uniformly random U and V in $\mathbb{Z}_q^{i \times j}$ and $\mathbb{Z}_q^{j \times k}$, respectively, computes $W = UV$ and pre-distributes secret sharings $[\![U]\!]_q, [\![V]\!]_q, [\![W]\!]_q$ to Alice and Bob.
2. Alice and Bob locally compute $[\![D]\!]_q \leftarrow [\![X]\!]_q - [\![U]\!]_q$ and $[\![E]\!]_q \leftarrow [\![Y]\!]_q - [\![V]\!]_q$, and then open D and E.
3. Alice and Bob locally compute $[\![Z]\!]_q \leftarrow [\![W]\!]_q + E[\![U]\!]_q + D[\![V]\!]_q + DE$.

Converting to Fixed-Point Representation

Each data owner initially needs to convert their training data to integers modulo q so that they can be secret shared. As illustrated in Figure 6, each feature value $x \in \mathbb{R}$ is converted into a fixed point approximation of x using a two's complement

representation for negative numbers. We define this new value as $Q(x) \in \mathbb{Z}_q$. This conversion is shown in Equation (1):

$$Q(x) = \begin{cases} 2^\lambda - \lfloor 2^a \cdot |x| \rfloor & \text{if } x < 0 \\ \lfloor 2^a \cdot x \rfloor & \text{if } x \geq 0 \end{cases} \tag{1}$$

Specifically, when we convert $Q(x)$ into its bit representation, we define the first a bits from the right to hold the fractional part of x, and the next b bits to represent the non-negative integer part of x, and the most significant bit (MSB) to represent the sign (positive or negative). We define λ to represent the total number of bits such that the ring size q is defined as $q = 2^\lambda$. It is important to choose a λ that is large enough to represent the maximum value produced during the LR protocol, and therefore it should be chosen to be at least $2(a+b)$. It is also important to choose a b that is large enough to represent the maximum possible value of the integer part of all x's (this is dependent on the data). This conversion and bit representation is shown in Figure 6.

Bit-Decomposition

We use the two-party protocol from [15] for performing the bit-decomposition of a secret-shared value $[\![x]\!]_{2^\lambda}$ to shares $[\![x_i]\!]_2$, where $x_\lambda \cdots x_1$ is the binary representation of x. It works like the ripple carry adder arithmetic circuit based on the insight that the difference between the sum of two additive shares and an "XOR-sharing" of that sum is the carry vector. As proven in [15], the bit-decomposition protocol π_{decomp} described below UC-realizes the bit-decomposition functionality $\mathcal{F}_{\mathsf{decomp}}$ described in Figure 3.

Protocol 2: Secure Two-Party Bit-Decomposition Protocol π_{decomp}

Input : $[\![x]\!]_{2^\lambda}$

Output: $[\![x_i]\!]_2$, where $x_\lambda \cdots x_1$ is the binary representation of x.

1 All distributed multiplications are over $\mathbb{Z}_2$ and the required correlated randomness is pre-distributed by the trusted initializer.

2 Let a denote Alice's share of x, which corresponds to the bit string $a_\lambda \ldots a_1$. Similarly, let b denote Bob's share of x, which corresponds to the bit string $b_\lambda \ldots b_1$. Define the secret sharings $[\![y_i]\!]_2$ as the pair of shares (a_i, b_i) for $y_i = a_i + b_i \mod 2$, $[\![a_i]\!]_2$ as $(a_i, 0)$ and $[\![b_i]\!]_2$ as $(0, b_i)$.

3 Compute $[\![c_1]\!]_2 \leftarrow [\![a_1]\!]_2 [\![b_1]\!]_2$ and set $[\![x_1]\!]_2 \leftarrow [\![y_1]\!]_2$.

4 **for** $i \leftarrow 2$ **to** λ **do**

5 Compute $[\![d_i]\!]_2 \leftarrow [\![a_i]\!]_2 [\![b_i]\!]_2 + 1$.

6 $[\![e_i]\!]_2 \leftarrow [\![y_i]\!]_2 [\![c_{i-1}]\!]_2 + 1$

7 $[\![c_i]\!]_2 \leftarrow [\![e_i]\!]_2 [\![d_i]\!]_2 + 1$

8 $[\![x_i]\!]_2 \leftarrow [\![y_i]\!]_2 + [\![c_{i-1}]\!]_2$

9 **end**

10 Output $[\![x_i]\!]_2$ for $i \in \{1, \ldots, \lambda\}$.

We have implemented a highly parallelized version of the bit-decomposition protocol named batch decomposition described in the methods section.

Functionality $\mathcal{F}_{\text{decomp}}$

$\mathcal{F}_{\text{decomp}}$ runs with Alice and Bob and is parametrized by the bit-length λ of the value x being converted from additive sharings $[\![x]\!]_{2^\lambda}$ in $\mathbb{Z}_{2^\lambda}$ to additive bitwise sharings $[\![x_i]\!]_2$ in $\mathbb{Z}_2$ such that $x = x_\lambda \cdots x_1$.

Input: Upon receiving a message from Alice or Bob with its share of $[\![x]\!]_q$, record the share, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct the value $x = x_\lambda \cdots x_1$ from the shares, and for $i \in \{1, \ldots, \lambda\}$ distribute new sharings $[\![x_i]\!]_2$ of the bit x_i. Before the output deliver, the corrupt party fix its shares of the output to any desired value. The shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraints.

Figure 3 The bit-decomposition functionality.

Protocol 3: Batch Bit Decomposition (batch-π_{decomp})
Constraints: Each of the n values are shares over $\mathbb{Z}_{2^\lambda}$

Input : X, list of n values shared over $\mathbb{Z}_{2^\lambda}$
Output: X', where $d'_i = \pi_{decomp}(x_i)$

Let A $\leftarrow$ X if Party 1 else {0,...,0}
Let B $\leftarrow$ {0,...,0} if Party 1 else X
Let AB $\leftarrow \{A_i \wedge B_i\}$ for $i \in \{1...n\}$
Let Y $\leftarrow$ AB
Let X' $\leftarrow \{Y_{i1}\}$ for $i \in \{1...n\}$
Let D $\leftarrow \{\overline{AB_i}\}$ for $i \in \{1...n\}$
Let $c_{slice} \leftarrow \{\{AB_{i1}\}$ for $i \in \{1...n\}\}^T$
for $i \leftarrow 2$ **to** λ **do**
    Let $y_{slice} \leftarrow \{\{Y_{ji}\}$ for $j \in \{1...n\}\}^T$
    Let $d_{slice} \leftarrow \{\{D_{ji}\}$ for $j \in \{1...n\}\}^T$
    Let $e_{slice} \leftarrow y_{slice} \wedge c_{slice}$
    for $j \leftarrow 1$ **to** n **do**
        $X'_{ij} \leftarrow (y_{slice,j} \oplus c_{slice,j}) << i$
    end
    $c_{slice} \leftarrow d_{slice} \wedge e_{slice}$
end
return X'

Bit Sharing to Additive Sharing

The opposite of a secure bit decomposition is converting from a bit sharing to additive sharing. In our secure activation function protocol π_ρ, we require securely

converting a number from its bit decomposition to an additive sharing (notated as π^{-1}_{decomp}). This is done as shown in the following steps:

1 Both parties have shares of a number x that is shared bitwise as $[\![b_{\lambda-1}, ..., b_0]\!]_2$, where λ is the number of bits. To convert additive sharings over the ring $\mathbb{Z}_2$ to sharings over the ring $\mathbb{Z}_{2^\lambda}$, we use the protocol π_{2toQ} from [16]. This is done for every bit b_i, and results in $[\![b_{\lambda-1}, ..., b_0]\!]_{2^\lambda}$.

2 Each bit is then multiplied by the corresponding power of 2 and added together (similar to converting a binary number to a decimal number): $[\![x]\!]_{2^\lambda} = \sum_{i=0}^{\lambda-1} [\![b_i]\!] 2^i$

Secure Activation Function

We propose a new protocol that evaluates ρ directly over additive shares and does not require full secure comparisons, which would have incurred more expensive secure multiplications. In other words, instead of doing a straightforward comparison between z, 0.5 and -0.5, we derive the result through checking two things: (i) whether z is positive or negative, and (ii) whether $|z| \geq 0.5$, both of which can be determined without using a full comparison protocol. When z is bit decomposed, the most significant bit (MSB) is 0 if x is positive and 1 if x is negative. When $|z|$ is bit decomposed, we can determine if $|z| \geq 0.5$ by just looking at the b integer bits and the most significant a fractional bits. It follows that $|z| \geq 0.5$ if and only if at least one of those bits is 1. We use the notation $c = 1$ if there is at least one bit that is set to 1, and $c = 0$ if all of those bits are 0. Using only this information, we can map z to any of the three regions of ρ as follows:

- if $\mathsf{MSB} = 1$ and $c = 1$, then $\rho(z) = 0$
- if $\mathsf{MSB} = 0$ or 1 and $c = 0$, then $\rho(z) = z + 0.5$
- if $\mathsf{MSB} = 0$ and $c = 1$, then $\rho(z) = 1$

To compute this securely, we use the following secure protocols as building blocks. We use the secure two-party bit-decomposition protocol π_{decomp} from [15] that converts additive sharings over a ring $\mathbb{Z}_{2^n}$ into n shares over the ring $\mathbb{Z}_2$. To convert additive sharings over the ring $\mathbb{Z}_2$ to sharings over a ring $\mathbb{Z}_{2^n}$, we use the protocol π_{2toQ} from [16].

This secure protocol is described in Algorithm 4, and we provide a detailed explanation here:

1 Store the most significant bit of z shared over 2^λ as $[\![\mathsf{MSB}]\!]$.

2 Store $\frac{1}{2}$ as a value in $\mathbb{Z}_{2^\lambda}$ as $[\![\frac{1}{2}]\!]_{2^\lambda}$

3 We obtain a secret sharing $[\![z_{rct}]\!]_{2^\lambda}$ corresponding to the absolute value of z by assigning $[\![z_{rct}]\!]_{2^\lambda} \leftarrow [\![z]\!]_{2^\lambda}$ if z is positive, and assigning $[\![z_{rct}]\!]_{2^\lambda} \leftarrow 2^\lambda - [\![z]\!]_{2^\lambda}$ if z is negative.

4 $[\![z_{rct}]\!]_{2^\lambda}$ is bit-decomposed using π_{decomp}, which results in shares of each bit over $\mathbb{Z}_2$ notated as $[\![e_\lambda, ..., e_1]\!]_2$.

5 Store only the bits of $[\![e_\lambda, ..., e_1]\!]_2$ that are greater than or equal to $\frac{1}{2}$ as $[\![d_{b+1}, ..., d_1]\!]_2$: this corresponds to the b integer bits and the most significant bit of the a fractional bits.

6 Obtain only the bits of $[\![e_\lambda, ..., e_1]\!]_2$ that are less than $\frac{1}{2}$: this corresponds to the $a - 1$ lowest bits. Then, convert those bit shares over $\mathbb{Z}_2$ to one secret sharing over $\mathbb{Z}_{2^\lambda}$

7 The bitwise OR of $[\![d_{b+1}, ..., d_1]\!]_2$ is logically equivalent to $|z| \geq \frac{1}{2}$. By De Morgan's law, this bitwise OR can be computed by negating the bitwise AND of the negated bits (where bitwise AND is equivalent to bitwise multiplication). The result, $[\![c]\!]_2$, determines whether $\rho(z)$ is in the constant or linear region.
8 Convert the sharing $[\![c]\!]_2$ to be a sharing over $\mathbb{Z}_{2^\lambda}$ ($[\![c]\!]_{2^\lambda}$
9 We let $[\![r_{temp}]\!]_{2^\lambda}$ be $[\![\frac{1}{2}]\!]_{2^\lambda}$ if $\rho(z)$ is in the constant region, or $[\![c]\!]_{2^\lambda}$ if $\rho(z)$ is in the linear region.
10 The final result is computed as being $[\![r_{temp}]\!]_{2^\lambda} + [\![\frac{1}{2}]\!]$ if z is positive, or $2^\lambda - [\![r_{temp}]\!]_{2^\lambda} + [\![\frac{1}{2}]\!]$ if z is negative.

Protocol 4: Secure Activation Function ρ over the ring $\mathbb{Z}_{2^\lambda}(\pi_\rho)$
Constraints: all values in $\mathbb{Z}_{2^\lambda}$ are representations of fixed point approximations of real numbers s.t. the lowest a bits represent the fractional component, the next b bits represent the integer component and $\lambda \geq 2(a+b)$. Further, a negative value x is represented as $2^\lambda - |x|$, where $|\cdot|$ is the absolute value of an integer.

Input : $[\![z]\!]_{2^\lambda}$
Output: $[\![\rho(z)]\!]_{2^\lambda}$
1 Let $[\![\mathsf{MSB}]\!]_{2^\lambda} \leftarrow \pi_{2toQ}(\pi_{decomp}([\![z]\!]_{2^\lambda}) >> \lambda - 1)$
2 Let $[\![\frac{1}{2}]\!]_{2^\lambda} \leftarrow 1 << (a-1)$
3 Let $[\![z_{rct}]\!]_{2^\lambda} \leftarrow (1 - [\![\mathsf{MSB}]\!]_{2^\lambda}) \cdot [\![z]\!]_{2^\lambda} + [\![\mathsf{MSB}]\!]_{2^\lambda} \cdot (2^\lambda - [\![z]\!]_{2^\lambda})$
4 Let $[\![e_\lambda, ..., e_1]\!]_2 \leftarrow \pi_{decomp}([\![z_{rct}]\!]_{2^\lambda})$ where $e_i \in \{0, 1\}$
5 Let $[\![d_{b+1}, ..., d_1]\!]_2 \leftarrow ([\![e_\lambda, ..., e_1]\!]_2 >> a - 1) \wedge (2^{b+1} - 1)$ where $d_i \in \{0, 1\}$
6 Let $[\![z_{frac}]\!]_{2^\lambda} \leftarrow \pi^{-1}_{decomp}([\![e_\lambda, ..., e_1]\!]_2 \wedge (2^{a-1} - 1))$
7 Let $[\![c]\!]_2 \leftarrow \neg \left(\bigwedge_{i=1}^{b+1} \overline{d_i} \right)$
8 Let $[\![c]\!]_{2^\lambda} \leftarrow \pi_{2toQ}([\![c]\!]_2)$
9 Let $[\![r_{temp}]\!]_{2^\lambda} \leftarrow [\![c]\!]_{2^\lambda} \cdot [\![\frac{1}{2}]\!]_{2^\lambda} + (1 - [\![c]\!]_{2^\lambda}) \cdot [\![z_{frac}]\!]_{2^\lambda}$
10 Let $[\![r]\!]_{2^\lambda} \leftarrow [\![\frac{1}{2}]\!]_{2^\lambda} + (1 - [\![\mathsf{MSB}]\!]_{2^\lambda}) \cdot [\![r_{temp}]\!]_{2^\lambda} + [\![\mathsf{MSB}]\!]_{2^\lambda} \cdot (2^\lambda - [\![r_{temp}]\!]_{2^\lambda})$
11 **return** $[\![r]\!]_{2^\lambda}$

Truncation

When multiplying numbers that were converted into a fixed point representation with a fractional bits, the resulting product will end up with a more bits representing the fractional part. For example, a fixed point representation of x and y is $x \cdot 2^a$ and $y \cdot 2^a$ respectively. The multiplication of both these terms results in $xy \cdot 2^{2a}$, showing that now $2a$ bits are representing the fractional part, which we must scale back down to $xy \cdot 2^a$ to do any further computations. In our solution, we use the two-party offline truncation protocol for fixed point representations of real numbers proposed in [6] that we will refer to as π_{trunc}. This protocol always incurs an error of at most a bit flip in the least-significant bit. However, with probability $2^{a+1-\lambda}$, where a is the number of fractional bits, the resulting value is completely random. When this truncation protocol is performed on increasingly large data sets (in our case we run over 7 billion secure multiplications), the probability of an erroneous truncation becomes a real issue - an issue not significant in previous implementations. There are two phases in which truncation is performed: (1) when computing the dot product of the current weights vector with a training example,

and (2) when the weights are updated at the end of a round. If a truncation error occurs during (1), the resulting erroneous value will be pushed into a reasonable range by the activation function and incur only a minor error for that round. If the error occurs during (2), an element of the weights vector will be updated to a completely random ring element and recovery from this error will be impossible. To mitigate this in experiments, we make use of 10-12 bits of fractional precision with a ring size of 64 bits, making the probability of failure $\frac{1}{2^{53}} < p < \frac{1}{2^{51}}$. The number of truncations that need to be performed was also reduced in our implementation by waiting to perform truncation until it is absolutely required. For instance, instead of truncating each result of multiplication between an attribute and its corresponding weight, a single truncation can be performed at the end of the entire dot product. Additional error is incurred on the accuracy by the fixed point representation itself. Through cross-validation with an in the clear implementation, we determined that 12 bits of fractional precision provide enough accuracy to make the output accuracy indistinguishable.

Logistic Regression Training in the Clear

Logistic regression is a common machine learning algorithm for binary classification. The training data D consists of training examples $d = (\boldsymbol{x}_d, t_d)$ in which $\boldsymbol{x}_d = \langle x_{d,1}, x_{d,2}, ..., x_{d,m} \rangle$ is an m-dimensional numerical vector, containing the values of m input attributes for example d, and $t_d \in \{0, 1\}$ is the ground truth class label. Each $x_{d,i}$ for $i \in \{1, 2, ..., m\}$ is a real number value. As illustrated in Figure 5(a), we train a neuron to map the $\boldsymbol{x}_d$'s to the corresponding t_d's, correctly classifying the examples. The neuron computes a weighted sum of the inputs (the values of the weights are learned during training) and subsequently applies an activation function to it, to arrive at the output $o_d = f(w_0 \cdot x_{d,0} + w_1 \cdot x_{d,1} + \cdots + w_n \cdot x_{d,n})$. Note that, as is common in neural network training, we extend the input attribute vector with a dummy feature $x_{d,0}$ which has value 1 for all $\boldsymbol{x}_d$'s. The traditionally used activation function for LR is the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$. Since the sigmoid function σ requires division and evaluation of an exponential function, which are expensive operations to perform in secret sharing based MPC, we approximate it with function ρ as from [6]. The function ρ is shown in Figure 5(b) and Equation 2.

$$\rho(z) = \begin{cases} 0 & \text{if } z < -\frac{1}{2} \\ z + \frac{1}{2} & \text{if } -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1 & \text{if } z > \frac{1}{2} \end{cases} \tag{2}$$

For training, we use the gradient descent based algorithm shown in Algorithm 5 to learn the weights for the LR model. On line 3, we choose not to use *early stopping*[2] because in that case the number of iterations would depend on the values in the training data, hence leaking information [7]. Instead, we use a fixed number of iterations to perform during training.

[2]This is a technique that uses a metric, such as a validation set's accuracy, to check when a model starts to overfit and will then stop training at that point.

Protocol 5: Full Gradient Descent

Input : A set D with training examples $(\boldsymbol{x}_d, t_d)$; a learning rate η

Output: Weights w_i that minimize the sum of squared errors over the training data

1 **for** $i \leftarrow 0$ **to** m **do**
2 $\quad w_i \leftarrow 0$
3 **until** *termination condition is met* **do**
4 $\quad$ **for** $i \leftarrow 0$ **to** m **do**
5 $\quad\quad \Delta w_i \leftarrow 0$
6 $\quad$ **for each** $(\boldsymbol{x}_d, t_d)$ *in* D **do**
7 $\quad\quad o_d \leftarrow \rho(w_0 \cdot x_{d,0} + w_1 \cdot x_{d,1} + \ldots + w_m \cdot x_{d,m})$
8 $\quad\quad$ **for** $i \leftarrow 0$ **to** m **do**
9 $\quad\quad\quad \Delta w_i \leftarrow \Delta w_i + \eta(t_d - o_d)x_{d,i}$
10 $\quad$ **for** $i \leftarrow 0$ **to** m **do**
11 $\quad\quad w_i \leftarrow w_i + \Delta w_i$

Secure Logistic Regression Training

We now present our secure LR training protocol that uses a combination of the previously mentioned building blocks.

Notice that in the gradient descent Algorithm 5, the only operations that cannot be performed fully locally by the data processors, i.e. on their own local shares, are:

- The computation of the inner product in line 7
- The activation function ρ in line 7
- The multiplication of $t_d - o_d$ with $d_{d,i}$ in line 9

In Algorithm 6, we show how we use the secure building blocks to securely compute these operations. The inner product is securely computed using π_{IP} on line 5, and since this involves multiplication on numbers that are scaled to a fixed-point representation, we truncate the result using π_{trunc} on line 6. The activation function is securely computed using π_ρ on line 8. The multiplication of $t_d - o_d$ with $x_{d,i}$ is done using batch-π_{DM} on line 13. Since this also involves multiplication on numbers that are scaled, the result is truncated using π_{trunc} in line 16.

A slight difference in the gradient descent Algorithm 5 and the secure training Algorithm 6 is instead of updating Δw_i after every evaluation of the activation function, we batch together all activation function evaluations before computing the Δw_i. Since the activation function requires a bit decomposition of the input, we can now make use of the efficient batch bit decomposition protocol batch-π_{decomp} within the activation function protocol π_ρ.

Protocol 6: Secure Logistic Regression Training

Input : A set D with training examples $([\![\boldsymbol{x}_d]\!], [\![t_d]\!])$; a learning rate η; number of iterations n_{iter}

Output: Weights $[\![w_i]\!]$ that minimize the sum of squared errors over the training data

for $i \leftarrow 0$ **to** m **do**
  $[\![w_i]\!] \leftarrow 0$
for 0 **to** n_{iter} **do**
  for each $(\boldsymbol{x}_d, t_d)$ *in* D **do**
    $[\![z_d]\!] \leftarrow \pi_{IP}([\![\langle w_0, w_1, ..., w_m \rangle]\!], [\![\langle x_{d,0}, x_{d,1}, ..., x_{d,m} \rangle]\!])$
    $[\![z_d]\!] \leftarrow \pi_{trunc}([\![z_d]\!])$
  for each $(\boldsymbol{x}_d, t_d)$ *in* D **do**
    $o_d \leftarrow \pi_{\rho}([\![z_d]\!])$
  for $i \leftarrow 0$ **to** m **do**
    $[\![\Delta w_i]\!] \leftarrow 0$
  for each $(\boldsymbol{x}_d, t_d)$ *in* D **do**
    $[\![\boldsymbol{v}_{diff}]\!] \leftarrow \langle ([\![t_d]\!] - [\![o_d]\!]), ..., ([\![t_d]\!] - [\![o_d]\!]) \rangle$ // vector of length m
    $[\![\boldsymbol{v}_{gradient}]\!] \leftarrow$batch-$\pi_{DM}([\![\boldsymbol{v}_{diff}]\!], [\![\boldsymbol{x}_d]\!])$
    $[\![\Delta \boldsymbol{w}]\!] \leftarrow [\![\Delta \boldsymbol{w}]\!] + [\![\boldsymbol{v}_{gradient}]\!]$
  for $i \leftarrow 0$ **to** m **do**
    $[\![\Delta w_i]\!] \leftarrow \pi_{trunc}([\![\Delta w_i]\!])$
  for $i \leftarrow 0$ **to** m **do**
    $[\![\Delta w_i]\!] \leftarrow \eta \cdot [\![\Delta w_i]\!]$
  for $i \leftarrow 0$ **to** m **do**
    $[\![w_i]\!] \leftarrow [\![w_i]\!] + [\![\Delta w_i]\!]$

Overview of all steps

Our setup is illustrated in Figure 4 where we have multiple data owners who each hold onto one training example $d = (\boldsymbol{x}_d, t_d)$, two data processors who collaborate in the training using the secure MPC protocols, and a TI that predistributes correlated randomness to the data processors for the efficient secure multiplication. The TI is not involved in any other part of the execution, and does not learn any data from the data owners or data processors.

The following steps describe end-to-end how to securely train a LR classifier:

1 Each data owner converts the values in their training example $d = (\boldsymbol{x}_d, t_d)$ to a fixed-point representation as described in Equation 1. Each value is then split into two shares, which are then sent to the data processor 1 and data processor 2 as $([\![\boldsymbol{x}_{d,p1}]\!], [\![t_{d,p1}]\!])$ and $([\![\boldsymbol{x}_{d,p2}]\!], [\![t_{d,p2}]\!])$ respectively.

2 The TI starts sending the correlated randomness needed for efficient secure multiplication to the data processors. Note that while our current implementation has the TI continuously sending the correlated randomness, it is possible for the TI to send all correlated randomness as the first step, and therefore can leave and not be involved during the rest of the protocol.

3 Each data processor receives the shares of data from the data owners. They now have a set D of shares of training examples $([\![\boldsymbol{x}_d]\!], [\![t_d]\!])$. The learning rate

η and number of iterations n_{iter} is predetermined and public to both data processors.

4 The data processors collaborate to train the LR classifier on shares of the training examples. They both follow the secure protocol in Algorithm 6.

5 At the end of the protocol, each data processor will hold shares of the model's weights $[\![w_i]\!]$. Each data processor sends their shares to all of the data owners, who can then combine the shares ($[\![w_{i,p1}]\!]$+$[\![w_{i,p2}]\!]$)to reveal the learned weights.

Cryptographic Engineering Optimizations

Sockets and Threading

A single iteration of the LR protocol is highly parallelizeble in three distinct segments: (1) computing the dot products between the current weights and the data set, (2) computing the activation of each dot product result, and (3) computing the gradient and updating the weights. In each of these phases, a large number of computations are required, but none have dependencies on others. We take advantage of this by completing each of these phases with thread pools that can be configured for the machine running the protocol. With Rust's ownership concept, it is possible to yield results from threads without message passing or reallocation. Hence, the code is constructed to transfer ownership of results at each phase back to the main thread to avoid as much inter-process communication as possible. Additionally, all threads complete socket communications by computing all intermediate results directly in the socket buffer by implementing the buffer as a union of byte array and unsigned 64-bit integer array. This buffer is allocated on the stack by each thread which circumvents the need for a shared memory block while also avoiding slower heap memory. The implementation of this configuration reduced running times significantly based on our trials. Further, all modular arithmetic operations are handled implicitly with the Rust API's Wrapping struct which tells the ALU to ignore integer overflow. As long as the size of the ring over which the MPC protocols are performed is selected to align with a provided primitive bit width (i.e. 8, 16, 32, 64, 128) it is possible to omit computing the remainder of arithmetic with this construction.

Batching of π_{decomp}

The fact that the carry bit of each bitwise sum must be present for the computation of the next bit implies a communication cost that is linear with the bit length λ. This is an unacceptable cost to pay when many bit decompositions need to take place. We propose batch-π_{decomp} as a means to take advantage of cases where many values need to be decomposed at once, as is the case in our LR protocol. It works by taking vertical slices of a collection of additive-shared values and transposing them into new integers. In this way, the carry of all 0-th order bits is computed, followed by all 1-st order bit carries, and so on up to the $(\lambda - 1)$-th carry. In this scheme, every group of λ ring elements requires only two sets of Beaver triples to compute the carries for the i-th bit. So until the optimal transmission buffer size is reached (based on our testing, 1024 bytes), the communication cost of performing multiple (4 times the transmission buffer size) bit decompositions is identical to only performing one. Additionally, the local computations are minimal and contribute virtually nothing to the running time.

Results

We implemented the protocols from the methods section in Rust[3] and experimentally evaluated them on the BC-TCGA and GSE2034 datasets of the iDASH2019 competition. Both datasets contain gene expression data from breast cancer patients which are normal tissue/non-recurrence samples (negative) or breast cancer tissue/recurrence tumor samples (positive) [25]. We trained LR models on both datasets with a learning rate $\eta = 0.001$. We use a fixed number of iterations for each dataset: 10 iterations for the BC-TCGA dataset and 223 iterations for the GSE2034 dataset. The accuracy of the resulting models, evaluated with 5-fold cross-validation is presented in Table 1, along with the average runtime (in min) for training those models. It is important to note that these are the same accuracies that are obtained when training in the clear, i.e. there is *no accuracy loss* in the secure version.

We used integer precision $b = 15$, fractional precision $a = 12$ and ring size $\lambda = 64$ (these choices were made based on experiments in the clear as mentioned in the section **Methods**:*Building Blocks:Truncation*). We ran the experiments on AWS c5.9xlarge machines with 36 vCPUs, 72.0 GiB Memory. Each of the parties ran on separate machines (connected with a Gigabit Ethernet network), which means that the results in Table 1 cover communication time in addition to computation time. The results show that our implementation allows to securely train models with state-of-the-art accuracy [25] on the BC-TCGA and GSE2034 datasets within about 2.71 seconds and 29.61 seconds respectively.

This implementation was submitted to the iDASH2019 Track 4 competition and won first place out of nine teams who also entered the competition. Our implementation trained on all of the features for both datasets (no feature engineering is done), and generated a model that gave the highest accuracy, with runtimes that were well within the competition's limit of 24 hours.

We note that while SecureML differs from our work in their setup and cryptographic primitives, it shares many similarities to ours and reports a fast runtime such that we find it valuable as a standard to compare to. We only compare online runtimes due to the differences in the offline phase. Specifically, while SecureML does not use a TI like us, it would be easy to use either method depending on the presence of a third party to act as a TI. We evaluated our implementation's runtime against SecureML's implementation by running their implementation on the same AWS machines using the same datasets (see Table 2 for runtime comparisons). For both datasets, our online phase runs faster than SecureML's online phase which trains BC-TCGA in 12.73 seconds and GSE2034 in 49.95 seconds.

We then compare online microbenchmark computation times. For the computation of the activation function, our run of the SecureML code reported around 0.057 ms to 0.059 ms for 1 activation, while our implementation completes 1024 evaluations in around 40.66 ms (around 0.039 ms per activation function). This makes our secure activation function implementation faster than SecureML's by 60%. Additionally, it eliminates the overhead of switching between Yao gates and additive secret sharing. Furthermore, our activation function runs more efficiently (per evaluation) the more evaluations of it need to be computed, due to the design of the batch bit decomposition protocol. This is illustrated in Table 3 where the calculated

[3]https://bitbucket.org/mdecock/idash2019-rust/

runtime per evaluation (runtime divided by number of evaluations) decreases as the number of evaluations increase.

Discussion

Our runtime experiments on securely training a LR model show that it is feasible to train on data that includes a large number of attributes, as is common with genomic data. Given the high dimensionality of the genomic data, an interesting direction for future work would be the design of MPC protocols for privacy-preserving feature reduction. If any kind of feature reduction is used, it would result in a decrease in secure training runtime with a possibility for a slight decrease in the accuracy. We demonstrate this by choosing (in the clear) 54 features of the BC-TCGA dataset that were part of the 76-gene signature described in [26]. Training on these 54 features, we get a 5 fold cross-validation accuracy of 98.93% (training on all features produced 99.58%), and the average secure training time (of three runs) is 0.78 seconds, which is about a 2 second decrease from training on all 17,814 features. The genes in the GSE2034 dataset are not labeled in a way where we can map them to the 76-gene signature to test the accuracy for a reduced number of features, but we test the runtime of training on 76 attributes and we get an average of 11.89 seconds, which is about a 38 second decrease from training on all 12,634 features. This shows that if feature reduction can be performed, runtimes can be improved while still being able to produce an accurate trained model.

Our main contribution is the proposal of the fastest implementation and protocol for privacy-preserving training of logistic regression models. Our novelty points are the new protocol for privately evaluating the activation function ρ which can be computed using only additive shares and MPC protocols, without using a protocol for secure comparison. We use ρ as an approximation of the sigmoid function σ since that is what is traditionally used in LR training, but σ is also used as an activation function in neural networks. Therefore, our fast secure protocol for computing ρ can also result in faster neural network training. While training neural networks are out of the scope of this paper, we note that our results can be applicable to those types of ML models as well.

Conclusions

In this paper, we have described a novel protocol for implementing secure training of LR over distributed parties using MPC. Our protocol and implementation present several novel points and optimizations compared to existing work, including: (i) a novel protocol for computing the activation function that does not require secure comparisons; (ii) a novel way to bundle many instances of secure bit decomposition and secure computations of the activation function.

With our implementation, we can train on the BC-TCGA dataset with 17,814 features and 375 samples with 10 iterations in 2.71 seconds, and we can train on the GSE2034 dataset with 12,634 features and 179 samples with 223 iterations in 29.61 seconds. This implementation won first place at the iDASH2019 Track 4 competition when considering accuracy and runtime. Our solution is particularly efficient for LANs where we can perform 1024 secure computations of the activation function in about 40.66 ms. To the best of our knowledge, ours is the fastest protocol for

privately training logistic regression models over local area networks. Our solution won first place in the iDash 2019 competion.

Abbreviations
ML: Machine learning; MPC: Multi-party computation; LR: Logistic regression; PP: Privacy-preserving; UC: Universal composability; PPML: Privacy-preserving machine learning; MSB: Most significant bit; TI: Trusted initializer;

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Availability of data and materials
The genomic dataset was available upon request during the iDASH 2019 competition.

Competing interests
The authors declare that they have no competing interests.

Funding
Not applicable.

Author's contributions
All authors worked together on the overall design of the solution in the clear and in private. DR designed the new cryptographic protocols for secure batch bit decomposition and secure activation function. DR implemented the entire solution in the RUST programming language. DR was responsible for running the experiments of our work, and AT was responsible for running the experiments on SecureML. RD verified and wrote the functionality and security proofs of our protocols. JS provided in-the-clear model testing, and worked on the submission details to the iDASH competition. All authors discussed results and wrote the manuscript together. All authors have read and approved the manuscript.

Acknowledgements
Not applicable.

Author details
[1]School of Engineering and Technology, University of Washington Tacoma, 98402 Tacoma, WA, USA.
[2]Department of Computer Science, Bar-Ilan University, 5290002 Ramat-Gan, Israel.

References
1. Jagadeesh KA, Wu DJ, Birgmeier JA, Boneh D, Bejerano G. Deriving genomic diagnoses without revealing patient genomes. Science. 2017;357(6352):692–695.
2. Bonte C, Vercauteren F. Privacy-preserving logistic regression training. BMC Medical Genomics. 2018;11(4):86.
3. Chen H, Gilad-Bachrach R, Han K, Huang Z, Jalali A, Laine K, et al. Logistic regression over encrypted data from fully homomorphic encryption. BMC Medical Genomics. 2018;11(4):81.
4. Kim A, Song Y, Kim M, Lee K, Cheon JH. Logistic regression model training based on the approximate homomorphic encryption. BMC Medical Genomics. 2018;11(4):83.
5. El Emam K, Samet S, Arbuckle L, Tamblyn R, Earle C, Kantarcioglu M. A secure distributed logistic regression protocol for the detection of rare adverse drug events. Journal of the American Medical Informatics Association. 2012;20(3):453–461.
6. Mohassel P, Zhang Y. SecureML: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (SP); 2017. p. 19–38.
7. Nardi Y, Fienberg SE, Hall RJ. Achieving both valid and secure logistic regression analysis on aggregated data from different private sources. Journal of Privacy and Confidentiality. 2012;4(1).
8. Xie W, Wang Y, Boker SM, Brown DE. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. arXiv preprint arXiv:161101170. 2016;.
9. Wagh S, Gupta D, Chandran N. SecureNN: 3-Party Secure Computation for Neural Network Training. Proceedings on Privacy Enhancing Technologies. 2019;1:24.
10. Canetti R. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: 42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA; 2001. p. 136–145. Available from: https://doi.org/10.1109/SFCS.2001.959888.
11. Cramer R, Damgård I, Nielsen JB. Secure Multiparty Computation and Secret Sharing. Cambridge University Press; 2015.
12. De Cock M, Dowsley R, Nascimento ACA, Newman SC. Fast, Privacy Preserving Linear Regression over Distributed Datasets Based on Pre-Distributed Data. In: 8th ACM Workshop on Artificial Intelligence and Security (AISec); 2015. p. 3–14.
13. David B, Dowsley R, Katti R, Nascimento AC. Efficient unconditionally secure comparison and privacy preserving machine learning classification protocols. In: International Conference on Provable Security. Springer; 2015. p. 354–367.
14. Fritchman K, Saminathan K, Dowsley R, Hughes T, De Cock M, Nascimento A, et al. Privacy-Preserving Scoring of Tree Ensembles: A Novel Framework for AI in Healthcare. In: Proc. of 2018 IEEE International Conference on Big Data; 2018. p. 2412–2421.

15. De Cock M, Dowsley R, Horst C, Katti R, Nascimento A, Poon WS, et al. Efficient and Private Scoring of Decision Trees, Support Vector Machines and Logistic Regression Models based on Pre-Computation. IEEE Transactions on Dependable and Secure Computing. 2019;16(2):217–230.
16. Reich D, Todoki A, Dowsley R, De Cock M, Nascimento A. Privacy-Preserving Classification of Personal Text Messages with Secure Multi-Party Computation. In: Advances in Neural Information Processing Systems (NeurIPS); 2019. .
17. Rivest RL. Unconditionally Secure Commitment and Oblivious Transfer Schemes Using Private Channels and a Trusted Initializer; 1999. Preprint available at http://people.csail.mit.edu/rivest/Rivest- commitment.pdf.
18. Dowsley R, Van De Graaf J, Marques D, Nascimento AC. A two-party protocol with trusted initializer for computing the inner product. In: International Workshop on Information Security Applications. Springer; 2010. p. 337–350.
19. Dowsley R, Müller-Quade J, Otsuka A, Hanaoka G, Imai H, Nascimento ACA. Universally Composable and Statistically Secure Verifiable Secret Sharing Scheme Based on Pre-Distributed Data. IEICE Transactions. 2011;94-A(2):725–734.
20. Ishai Y, Kushilevitz E, Meldgaard S, Orlandi C, Paskin-Cherniavsky A. On the power of correlated randomness in secure computation. In: Theory of Cryptography. Springer; 2013. p. 600–620.
21. Tonicelli R, Nascimento ACA, Dowsley R, Müller-Quade J, Imai H, Hanaoka G, et al. Information-theoretically secure oblivious polynomial evaluation in the commodity-based model. International Journal of Information Security. 2015;14(1):73–84.
22. David B, Dowsley R, van de Graaf J, Marques D, Nascimento ACA, Pinto ACB. Unconditionally Secure, Universally Composable Privacy Preserving Linear Algebra. IEEE Transactions on Information Forensics and Security. 2016;11(1):59–73.
23. Beaver D. Commodity-based cryptography. In: STOC. vol. 97; 1997. p. 446–455.
24. Dowsley R. Cryptography Based on Correlated Data: Foundations and Practice. Karlsruhe Institute of Technology, Germany; 2016.
25. Xie H, Li J, Zhang Q, Wang Y. Comparison among dimensionality reduction techniques based on Random Projection for cancer classification. Computational Biology and Chemistry. 2016;65:165–172.
26. Wang Y, Klijn JG, Zhang Y, Sieuwerts AM, Look MP, Yang F, et al. Gene-expression profiles to predict distant metastasis of lymph-node-negative primary breast cancer. The Lancet. 2005;365(9460):671–679.

Figures

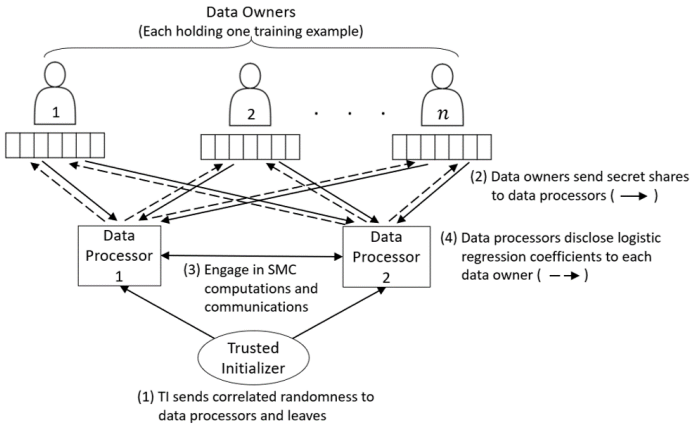


Figure 4 Overview of MPC based secure logistic regression (LR) training Each of n data owners secret shares their own training data between two data processors. The data processors engage in computations and communications to train a ML model, which is at the end revealed to the data owners.

Tables

Table 1 Accuracy and training runtime for LR models

	# features	# pos. samples	# neg. samples	# of of iterations	5-fold CV accuracy	avg. runtime
BC-TCGA	17,814	422	48	10	99.58%	2.71 sec
GSE2034	12,634	142	83	223	64.82%	29.61 sec

Additional Files

Additional file 1 — Figure 1

Image of Figure 1 Overview of MPC based secure logistic regression (LR) training

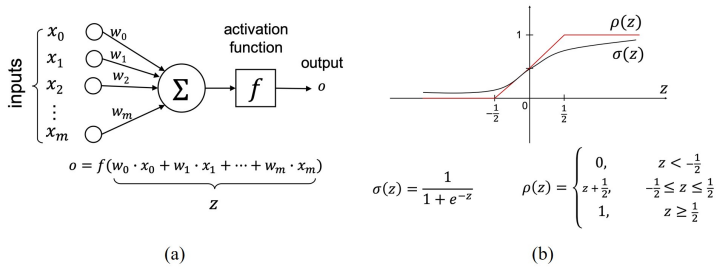


Figure 5 Architecture (a) Neuron; (b) Approximation of sigmoid activation function σ by clipped ReLu ρ

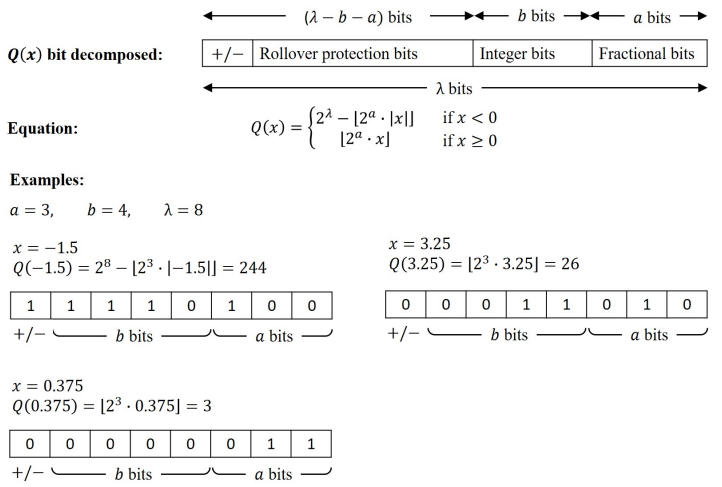


Figure 6 Fixed-Point Representation Register map of fixed-point representation of numbers shared over $\mathbb{Z}_{2^\lambda}$ with examples.

Table 2 Runtime comparisons between SeucreML and our work

	BC-TCGA training (online)	GSE2034 training (online)	activation function (one evaluation)
Our work	2.71 sec	29.61 sec	0.039 ms
SecureML	12.73 sec	49.95 sec	0.057 ms

Table 3 Activation function runtimes

# evaluations	avg. runtime	runtime per activation (runtime/#eval)
256	19.66 ms	0.076 ms
512	27 ms	0.052 ms
1024	40.66 ms	0.039 ms
2048	71 ms	0.034 ms

Additional file 2 — Figure 2
Image of Figure 2 Architecture (neuron and activation function)

Additional file 3 — Figure 3
Image of Figure 3 Fixed-Point Representation with examples of converting real numbers to the fixed-point representation.

Additional file 4 — Appendix
This file contains the security correctness and proofs for the protocols described in the paper.

Privacy-Preserving Classification of Personal Text Messages with Secure Multi-Party Computation: An Application to Hate-Speech Detection

Devin Reich[1], **Ariel Todoki**[1], **Rafael Dowsley**[2], **Martine De Cock**[1]*, **Anderson Nascimento**[1]
[1] School of Engineering and Technology
University of Washington Tacoma
Tacoma, WA 98402
{dreich,atodoki,mdecock,andclay}@uw.edu
[2]Department of Computer Science
Bar-Ilan University, 5290002, Ramat-Gan, Israel
rafael@dowsley.net

Abstract

Classification of personal text messages has many useful applications in surveillance, e-commerce, and mental health care, to name a few. Giving applications access to personal texts can easily lead to (un)intentional privacy violations. We propose the first privacy-preserving solution for text classification that is provably secure. Our method, which is based on Secure Multiparty Computation (SMC), encompasses both feature extraction from texts, and subsequent classification with logistic regression and tree ensembles. We prove that when using our secure text classification method, the application does not learn anything about the text, and the author of the text does not learn anything about the text classification model used by the application beyond what is given by the classification result itself. We perform end-to-end experiments with an application for detecting hate speech against women and immigrants, demonstrating excellent runtime results without loss of accuracy.

1 Introduction

The ability to elicit information through automated scanning of personal texts has significant economic and societal value. Machine learning (ML) models for classification of text such as e-mails and SMS messages can be used to infer whether the author is depressed [46], suicidal [42], a terrorist threat [1], or whether the e-mail is a spam message [2, 49]. Other valuable applications of text message classification include user profiling for tailored advertising [32], detection of hate speech [6], and detection of cyberbullying [51]. Some of the above are integrated in parental control applications[2] that monitor text messages on the phones of children and alert their parents when content related to drug use, sexting, suicide etc. is detected. Regardless of the clear benefits, giving applications access to one's personal text messages and e-mails can easily lead to (un)intentional privacy violations.

In this paper, we propose the first privacy-preserving (PP) solution for text classification that is provably secure. To the best of our knowledge, there are no existing Differential Privacy (DP) or Secure Multiparty Computation (SMC) based solutions for PP feature extraction and classification of unstructured texts; the only existing method is based on Homomorphic Encryption (HE) and takes 19 minutes to classify a tweet [15] while leaking information about the text being classified. In our SMC

*Guest Professor at Dept. of Applied Mathematics, Computer Science, and Statistics, Ghent University

[2]https://www.bark.us/, https://kidbridge.com/, https://www.webwatcher.com/

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

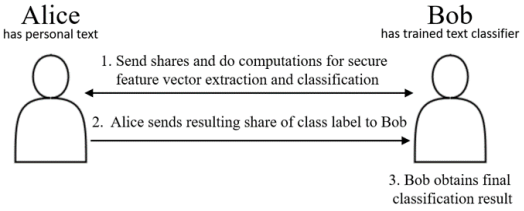


Figure 1: Roles of Alice and Bob in SMC based text classification

based solution, there are two parties, nick-named *Alice* and *Bob* (see Fig. 1). Bob has a trained ML model that can automatically classify texts. Our secure text classification protocol allows to classify a personal text written by Alice with Bob's ML model in such a way that Bob does not learn anything about Alice's text and Alice does not learn anything about Bob's model. Our solution relies on PP protocols for feature extraction from text and PP machine learning model scoring, which we propose in this paper.

We perform end-to-end experiments with an application for PP detection of hate speech against women and immigrants in text messages. In this use case, Bob has a trained logistic regression (LR) or AdaBoost model that flags hateful texts based on the occurrence of particular words. LR models on word n-grams have been observed to perform comparably to more complex CNN and LSTM model architectures for hate speech detection [35]. Using our protocols, Bob can label Alice's texts as hateful or not without learning which words occur in Alice's texts, and Alice does not learn which words are in Bob's hate speech lexicon, nor how these words are used in the classification process. Moreover, classification is done in seconds, which is two orders of magnitude better than the existing HE solution despite the fact we use over 20 times more features and do not leak any information about Alice's text to the model owner (Bob). The solution based on HE leaks which words in the text are present in Bob's lexicon [15].

We build our protocols using a privacy-preserving machine learning (PPML) framework based on SMC developed by us[3]. All the existing building blocks can be composed within themselves or with new protocols added to the framework. On top of existing building blocks, we also propose a novel protocol for binary classification over binary input features with an ensemble of decisions stumps. While some of our building blocks have been previously proposed, the main contribution of this work consists of the careful choice of the ML techniques, feature engineering and algorithmic and implementation optimizations to enable end-to-end practical PP text classification . Additionally, we provide security definitions and proofs for our proposed protocols.

2 Preliminaries

We consider *honest-but-curious adversaries*, as is common in SMC based PPML (see e.g. [19, 21]). An honest-but-curious adversary follows the instructions of the protocol, but tries to gather additional information. Secure protocols prevent the latter.

We perform SMC using additively secret shares to do computations modulo an integer q. A value x is secret shared over $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ between parties Alice and Bob by picking $x_A, x_B \in \mathbb{Z}_q$ uniformly at random subject to the constraint that $x = x_A + x_B \mod q$, and then revealing x_A to Alice and x_B to Bob. We denote this secret sharing by $[\![x]\!]_q$, which can be thought of as a shorthand for (x_A, x_B). Secret-sharing based SMC works by first having the parties split their respective *inputs* in secret shares and send some of these shares to each other. Naturally, these inputs have to be mapped appropriately to $\mathbb{Z}_q$. Next, Alice and Bob represent the *function* they want to compute securely as a circuit consisting of addition and multiplication gates. Alice and Bob will perform secure additions and multiplications, gate by gate, over the shares until the desired outcome is obtained. The final result can be recovered by combining the final shares, and disclosed as intended, i.e. to one of the parties or to both. It is also possible to keep the final result distributed over shares.

In SMC based text classification, as illustrated in Fig. 1, Alice's *input* is a personal text x and Bob's *input* is an ML model $\mathcal{M}$ for text classification. The function that they want to compute securely is

[3]https://bitbucket.org/uwtppml

$f(x, \mathcal{M}) = \mathcal{M}(x)$, i.e. the class label of x when classified by $\mathcal{M}$. To this end, Alice splits the text in secret shares while Bob splits the ML model in secret shares. Both parties engage in a protocol in which they send some of the input shares to each other, do local computations on the shares, and repeat this process in an iterative fashion over shares of intermediate results (Step 1). At the end of the joint computations, Alice sends her share of the computed class label to Bob (Step 2), who combines it with his share to learn the classification result (Step 3). As mentioned above, the protocol for Step 1 involves representing the function f as a circuit of addition and multiplication gates.

Given two secret sharings $[\![x]\!]_q$ and $[\![y]\!]_q$, Alice and Bob can locally compute in a straightforward way a secret sharing $[\![z]\!]_q$ corresponding to $z = x + y$ or $z = x - y$ by simply adding/subtracting their local shares of x and y modulo q. Given a constant c, they can also easily locally compute a secret sharing $[\![z]\!]_q$ corresponding to $z = cx$ or $z = x + c$: in the former case Alice and Bob just multiply their local shares of x by c; in the latter case Alice adds c to her share of x while Bob keeps his original share. These local operations will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + [\![y]\!]_q$, $[\![z]\!]_q \leftarrow [\![x]\!]_q - [\![y]\!]_q$, $[\![z]\!]_q \leftarrow c[\![x]\!]_q$ and $[\![z]\!]_q \leftarrow [\![x]\!]_q + c$, respectively. To allow for efficient secure multiplication of values via operations on their secret shares (denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q[\![y]\!]_q$), we use a trusted initializer that pre-distributes correlated randomness to the parties participating in the protocol before the start of Step 1 in Fig. 1.[4] The initializer is not involved in any other part of the execution and does not learn any data from the parties. This can be straightforwardly extended to efficiently perform secure multiplication of secret shared matrices. The protocol for secure multiplication of secret shared matrices is denoted by π_{DMM} and for the special case of inner-product computation by π_{IP}. Details about the (matrix) multiplication protocol can be found in [19]. We note that if a trusted initializer is not available or desired, Alice and Bob can engage in pre-computations to securely emulate the role of the trusted initializer, at the cost of introducing computational assumptions in the protocol [19].

3 Secure text classification

Our general protocol for PP text classification relies on several building blocks that are used together to accomplish Step 1 in Fig. 1: a secure equality test, a secure comparison test, private feature extraction, secure protocols for converting between secret sharing modulo 2 and modulo $q > 2$, and private classification protocols. Several of these building blocks have been proposed in the past. However, to the best of our knowledge, this is the very first time they are combined in order to achieve efficient text classification with provable security.

We assume that Alice has a personal text message, and that Bob has a LR or AdaBoost classifier that is trained on unigrams and bigrams as features. Alice constructs the set $A = \{a_1, a_2, \ldots, a_m\}$ of unigrams and bigrams occurring in her message, and Bob constructs the set $B = \{b_1, b_2, \ldots, b_n\}$ of unigrams and bigrams that occur as features in his ML model. We assume that all a_j and b_i are in the form of bit strings. To achieve this, Alice and Bob convert each unigram and bigram on their end to a number N using SHA 224 [44], strictly for its ability to map the same inputs to the same outputs in a pseudo-random manner. Next Alice and Bob map each N on their end to a number between 0 and $2^l - 1$, i.e. a bit string of length l, using a random function in the universal hash family proposed by Carter and Wegman [12].[5] In the remainder we use the term "word" to refer to a unigram or bigram, and we refer to the set $B = \{b_1, b_2, \ldots, b_n\}$ as Bob's lexicon.

Below we outline the protocols for PP text classification. A correctness and security analysis of the protocols is provided as an appendix. In the description of the protocols in this paper, we assume that Bob needs to learn the result of the classification, i.e. the class label, at the end of the computations. It is important to note that the protocols described below can be straightforwardly adjusted to a scenario where Alice instead of Bob has to learn the class label, or even to a scenario where neither Alice nor Bob should learn what the class label is and instead it should be revealed to a third party or kept in a secret sharing form. All these scenarios might be relevant use cases of PP text classification, depending on the specific application at hand.

[4] This technique for secure multiplication was originally proposed by Beaver [7] and is regularly used to enable very efficient solutions both in the context of PPML [20, 17, 33, 19] as well as in other applications, e.g., [48, 28, 27, 38, 50, 18].

[5] The hash function is defined as $((a \cdot N + b) \mod p) \mod 2^l - 1$ where p is a prime and a and b are random numbers less than p. In our experiments, $p = 1,301,081$, $a = 972$, and $b = 52,097$.

3.1 Cryptographic building blocks

Secure Equality Test: At the start of the secure equality test protocol, Alice and Bob have secret shares of two bit strings $x = x_\ell \dots x_1$ and $y = y_\ell \dots y_1$ of length ℓ. x corresponds to a word from Alice's message and y corresponds to a feature from Bob's model. The bit strings x and y are secret shared over $\mathbb{Z}_2$. Alice and Bob follow the protocol to determine whether $x = y$. The protocol π_{EQ} outputs a secret sharing of 1 if $x = y$ and of 0 otherwise.

Protocol π_{EQ}:
- For $i = 1, \dots, \ell$, Alice and Bob locally compute $[\![r_i]\!]_2 \leftarrow [\![x_i]\!]_2 + [\![y_i]\!]_2 + 1$.
- Alice and Bob use secure multiplication to compute a secret sharing of $z = r_1 \cdot r_2 \cdot \ldots \cdot r_\ell$. If $x = y$, then $r_i = 1$ for all bit positions i, hence $z = 1$; otherwise some $r_i = 0$ and therefore $z = 0$. The result is the secret sharing $[\![z]\!]_2$, which is the desired output of the protocol.

This protocol for equality test is folklore in the field of SMC. The $l - 1$ multiplications can be organized in as binary tree with the result of the multiplication at the root of the tree. In this way, the presented protocol has $\log(l)$ rounds. While there are equality test protocols that have a constant number of rounds, the constant is prohibitively large for the parameters used in our implementation.

Secure Feature Vector Extraction: At the start of the feature extraction protocol, Alice has a set $A = \{a_1, a_2, \dots, a_m\}$ and Bob has a set $B = \{b_1, b_2, \dots, b_n\}$. A is a set of bit strings that represent Alice's text, and B is a set of bit strings that represent Bob's lexicon. Bob would like to extract words from Alice's text that appear in his lexicon. At the end of the protocol, Alice and Bob have secret shares of a binary feature vector x which represents what words in Bob's lexicon appear in Alice's text. The binary feature vector x of length n is defined as

$$x_i = \begin{cases} 1 & \text{if } b_i \in A \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Protocol π_{FE}:
- Alice and Bob secret share each a_j ($j = 1, \dots, m$) and each b_i ($i = 1, \dots, n$) with each other.
- For $i = 1 \dots n$: // Computation of secret shares of x_i as defined in Equation (1).
  For $j = 1 \dots m$:
  Alice and Bob run the secure equality test protocol π_{EQ} to compute secret shares

$$x_{ij} = 1 \text{ if } a_j = b_i; \quad x_{ij} = 0 \text{ otherwise} \tag{2}$$

  Alice and Bob locally compute the secret share $[\![x_i]\!]_2 \leftarrow \sum_{j=1}^{m} [\![x_{ij}]\!]_2$.

The secure feature vector extraction can be seen as a private set intersection where the intersection is not revealed but shared [13, 31]. Our solution π_{FE} is tailored to be used within our PPML framework (it uses only binary operations, it is secret sharing based, and is based on pre-distributed binary multiplications). In principle, other protocols could be used here. The efficiency of our protocol can be improved by using hashing techniques [45] at the cost of introducing a small probability of error. The improvements due to hashing are asymptotic and for the parameters used in our fastest running protocol these improvements were not noticeable. Thus, we restricted ourselves to the original protocol without hashing and without any probability of failure.

Secure Comparison Test: In our privacy-preserving AdaBoost classifier we will use a secure comparison protocol as a building block. At the start of the secure comparison test protocol, Alice and Bob have secret shares over $\mathbb{Z}_2$ of two bit strings $x = x_\ell \dots x_1$ and $y = y_\ell \dots y_1$ of length ℓ. They run the secure comparison protocol π_{DC} of Garay et al. [34] with secret sharings over $\mathbb{Z}_2$ and obtain a secret sharing of 1 if $x \geq y$ and of 0 otherwise.

Secure Conversion between $\mathbb{Z}_q$ and $\mathbb{Z}_2$: Some of our building blocks perform computations using secret shares over $\mathbb{Z}_2$ (secure equality test, comparison and feature extraction), while the secure inner product works over $\mathbb{Z}_q$ for $q > 2$. In order to be able to integrate these building blocks we need:
- A secure bit-decomposition protocol for secure conversion from $\mathbb{Z}_q$ to $\mathbb{Z}_2$: Alice and Bob have as input a secret sharing $[\![x]\!]_q$ and without learning any information about x they should obtain as output secret sharings $[\![x_i]\!]_2$, where $x_\ell \cdots x_1$ is the binary representation of x. We use the secure bit-decomposition protocol π_{decomp} from De Cock et al. [19].

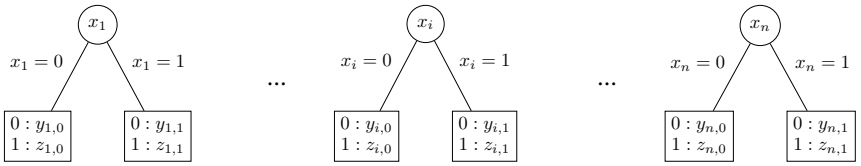


Figure 2: Ensemble of decision stumps. Each root corresponds to a feature x_i. The leaves contain weights $y_{i,k}$ for the votes for class label 0 and weights $z_{i,k}$ for the votes for class label 1.

- A protocol for secure conversion from $\mathbb{Z}_2$ to $\mathbb{Z}_q$: Alice and Bob have as a input a secret sharing $[\![x]\!]_2$ of a bit x and need to obtain a secret sharing $[\![x]\!]_q$ of the binary value over a larger field $\mathbb{Z}_q$ without learning any information about x. To this end, we use protocol π_{2toQ}:
  - For the input $[\![x]\!]_2$, let $x_A \in \{0,1\}$ denote Alice's share and $x_B \in \{0,1\}$ denote Bob's share.
  - Alice creates a secret sharing $[\![x_A]\!]_q$ by picking uniformly random shares that sum to x_A and delivers Bob's share to him, and Bob proceeds similarly to create $[\![x_B]\!]_q$.
  - Alice and Bob compute $[\![y]\!]_q \leftarrow [\![x_A]\!]_q [\![x_B]\!]_q$.
  - The output is computed as $[\![z]\!]_q \leftarrow [\![x_A]\!]_q + [\![x_B]\!]_q - 2[\![y]\!]_q$.

Secure Logistic Regression (LR) Classification: At the start of the secure LR classification protocol, Bob has a trained LR model $\mathcal{M}$ that requires a feature vector x of length n as its input, and produces a label $\mathcal{M}(x)$ as its output. Alice and Bob have secret shares of the feature vector x which represents what words in Bob's lexicon appear in Alice's text. At the end of the protocol, Bob gets the result of the classification $\mathcal{M}(x)$. We use an existing protocol π_{LR} for secure classification with LR models [19].[6]

Secure AdaBoost Classification: The setting is the same as above, but the model $\mathcal{M}$ is an AdaBoost ensemble of decision stumps instead of a LR model. While efficient solutions for secure classification with tree ensembles were previously known [33], we can take advantage of specific facts about our use case to obtain a more efficient solution. In more detail, in our use case: (1) all the decision trees have depth 1 (i.e., decision stumps); (2) each feature x_i is binary and therefore when it is used in a decision node, the left and right children correspond exactly to $x_i = 0$ and $x_i = 1$; (3) the output class is binary; (4) the feature values were extracted in a PP way and are secret shared so that no party alone knows their values. We can use the above facts in order to perform the AdaBoost classification by computing two inner products and then comparing their values.

Protocol π_{AB}:

- Alice and Bob hold secret sharings $[\![x_i]\!]_q$ of each of the n binary features x_i. Bob holds the trained AdaBoost model which consists of two weighted probability vectors $y = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1})$ and $z = (z_{1,0}, z_{1,1}, \ldots, z_{n,0}, z_{n,1})$. For the i-th decision stump: $y_{i,k}$ is the weighted probability (i.e., a probability multiplied by the weight of the i-th decision stump) that the model assigns to the output class being 0 if $x_i = k$, and $z_{i,k}$ is defined similarly for the output class 1 (see Fig. 2).
- Bob secret shares the elements of y and z, and Alice and Bob locally compute secret sharings $[\![w]\!]_q$ of the vector $w = (1 - x_1, x_1, 1 - x_2, x_2, \ldots, 1 - x_n, x_n)$.
- Using the secure inner product protocol π_{IP}, Alice and Bob compute secret sharings of the inner product p_0 between y and w, and of the inner product p_1 between z and w. p_0 and p_1 are the aggregated votes for class label 0 and 1 respectively.
- Alice and Bob use π_{decomp} to compute bitwise secret sharings of p_0 and p_1 over $\mathbb{Z}_2$.
- Alice and Bob use π_{DC} to compare p_1 and p_0, getting as output a secret sharing of the output class c, which is then open towards Bob.

To the best of our knowledge, this is the most efficient provably secure protocol for binary classification over binary input features with an ensemble of decisions stumps.

[6] In our case the result of the classification is disclosed to Bob (the party that owns the model) instead of Alice (who has the original input to be classified) as in [19], however it is trivial to modify their protocol so that the final secret share is open towards Bob instead of Alice. Note also that in our case, the feature vector that is used for the classification is already secret shared between Alice and Bob, while in their protocol Alice holds the feature vector, which is then secret shared in the first step of the protocol. This modification is also trivial and does not affect the security of the protocol.

Table 1: Accuracy (Acc) results using 5-fold cross-validation over the corpus of 10,000 tweets. Total time (Tot) needed to securely classify a text with our framework, broken down in time needed for feature vector extraction (Extr) and time for feature vector classification (Class).

	Unigrams				**Unigrams+Bigrams**			
	Acc	**Time (in sec)**			**Acc**	**Time (in sec)**		
		Extr	Class	Tot		Extr	Class	Tot
Ada; 50 trees; depth 1	71.6%	0.8	6.4	7.2	73.3%	1.5	6.6	8.1
Ada; 200 trees; depth 1	73.0%	2.8	6.4	9.2	74.2%	9.4	6.6	16.0
Ada; 500 trees; depth 1	73.9%	6.6	6.7	13.3	**74.4%**	**21.6**	**6.7**	**28.3**
Logistic regression (50 feat.)	**72.4%**	**0.8**	**3.7**	**4.5**	73.8%	1.5	3.8	5.3
Logistic regression (200 feat.)	73.3%	2.8	3.7	6.5	73.7%	9.4	3.8	13.2
Logistic regression (500 feat.)	73.4%	6.6	3.8	10.4	74.2%	21.6	4.1	25.7
Logistic regression (all feat.)	73.1%	318.0	6.1	324.1	73.8%	5,371.9	24.9	5,396.8

3.2 Privacy-preserving classification of personal text messages

We now present our novel protocols for PP text classification. They result from combining the cryptographic building blocks we introduced previously. The PP protocol $\pi_{\mathsf{TC-LR}}$ for classifying the text using a logistic regression model works as follows:

Protocol $\pi_{\mathsf{TC-LR}}$:
- Alice and Bob execute the secure feature extraction protocol π_{FE} with input sets A and B in order to obtain the secret shares $[\![x_i]\!]_2$ of the feature vector x.
- They run the protocol π_{2toQ} to obtain shares $[\![x_i]\!]_q$ over $\mathbb{Z}_q$.
- Alice and Bob run the secure logistic regression classification protocol π_{LR} in order to get the result of the classification. The LR model $\mathcal{M}$ is given as input to π_{LR} by Bob, and the secret shared feature vector x by both of them. Bob gets the result of the classification $\mathcal{M}(x)$.

The privacy-preserving protocol $\pi_{\mathsf{TC-AB}}$ for classifying the text using AdaBoost works as follows:

Protocol $\pi_{\mathsf{TC-AB}}$:
- Alice and Bob execute the secure feature extraction protocol π_{FE} with input sets A and B in order to obtain the secret shares $[\![x_i]\!]_2$ of the feature vector x.
- They run the protocol π_{2toQ} to obtain shares $[\![x_i]\!]_q$ over $\mathbb{Z}_q$.
- Alice and Bob run the secure AdaBoost classification protocol π_{AB} to obtain the result of the classification. The secret shared feature vector x is given as input to π_{AB} by both of them, and the two weighted probability vectors $y = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1})$ and $z = (z_{1,0}, z_{1,1}, \ldots, z_{n,0}, z_{n,1})$ that constitute the model are specified by Bob. Bob gets the output class c.

Detailed proofs of security are presented in the appendix.

4 Experimental results

We evaluate the proposed protocols in a use case for the detection of hate speech in short text messages, using data from [6]. The corpus consists of 10,000 tweets, 60% of which are annotated as hate speech against women or immigrants. We convert all characters to lowercase, and turn each tweet into a set of word unigrams and bigrams. There are 29,853 distinct unigrams and 93,629 distinct bigrams in the dataset, making for a total of 123,482 features.

Accuracy results for a variety of models trained to classify a tweet as hate speech vs. non-hate speech are presented in Table 1. The models are evaluated using 5-fold cross-validation over the entire corpus of 10,000 tweets. The top rows in Table 1 correspond to tree ensemble models consisting of 50, 200, and 500 decision stumps respectively; the root of each stump corresponds to a feature. The bottom rows contain results for an LR model trained on 50, 200, and 500 features (preselected based on information gain), and an LR model trained on all features. We ran experiments for feature sets consisting of unigrams and bigrams, as well as for feature sets consisting of unigrams only, observing that the inclusion of bigrams leads to a small improvement in accuracy. Note that designing a model to obtain the highest possible accuracy is not the focus of this paper. Instead, our goal is to demonstrate that PP text classification based on SMC is feasible in practice.

We implemented the protocols from Section 3 in Java and ran experiments on AWS c5.9xlarge machines with 36 vCPUs, 72.0 GiB Memory.[7] Each of the parties ran on separate machines (connected with a Gigabit Ethernet network), which means that the results in Table 1 cover communication time in addition to computation time. Each runtime experiment was repeated 3 times and average results are reported. In Table 1 we report the time (in sec) needed for converting a tweet into a feature vector (Extr), for classification of the feature vector (Class), and for the overall process (Tot).

4.1 Analysis

The best running times were obtained using unigrams, 50 features and logistic regression (4.5 s) with an accuracy of 72.4%. The highest accuracy (74.4%) was obtained by using unigram and bigrams, 500 features and AdaBoost with a running time equal to 28.3s. From these results, it is clear that feature engineering plays a major role in optimizing privacy-preserving machine learning solutions based on SMC. We managed to reduce the running time from 5,396.8s (logistic regression, unigram and bigrams, all 123,482 features being used) to 5.3s (logistic regression, unigrams and bigrams, 50 features) without any loss in accuracy and to 4.5s (logistic regression, unigrams only, 50 features) with a small loss.

4.2 Optimizing the computational and communication complexities

The feature extraction protocol requires $n \cdot m$ secure equality tests of bit strings. The equality test relies on secure multiplication, which is the more expensive operation. To reduce the number of required equality tests, Alice and Bob can each first map their bit strings to p buckets $A_1, A_2, \ldots, A_p$ and $B_1, B_2, \ldots, B_p$ respectively, so that bit strings from each A_i need to only be compared with bit strings from B_i. Each bit string a_j and b_i is hashed and the first t bits of the hash output are used to define the bucket number corresponding to that bit string, using a total of $p = 2^t$ buckets. In order not to leak how many elements are mapped to each bucket (which can leak some information about the probability distribution of the elements, as the hash function is known by everyone), each bucket has a fixed number of elements (s_1 for Bob's buckets and s_2 for Alice's buckets) and the empty spots in the buckets are filled up with dummy elements. The feature extraction protocol now requires $p \cdot s_1 \cdot s_2$ equality tests, which can be substantially smaller than $n \cdot m$. When using bucketization, the feature vector of length n from Equation (1) is expanded to a feature vector of length $p \cdot s_1$, containing the original n features as well as the $p \cdot s_1 - n$ dummy features that Bob created to fill up his buckets. These dummy features do not have any effect on the accuracy of the classification because Bob's model does not take them into account: the trees with dummy features in an AdaBoost model have 0 weight for both class labels, and the dummy features' coefficients in an LR model are always 0.

The size of the buckets has to be chosen sufficiently large to avoid overflow. The choice depends directly on the number $p = 2^t$ of buckets (which is kept constant for Alice and Bob) and the number of elements to be placed in the buckets, i.e. n elements on Bob's side and m elements on Alice's side. While for hash functions coming from a 2-universal family of hash functions the computation of these probabilities is relatively straightforward, the same is not true for more complicated hash functions [45]. In that case, numerical simulations are needed in order to bound the required probability.

The effect of using buckets is more significant for large values of n and m. In our case, after performing feature engineering for reducing the number of elements in each set, in the best case, we end up with inputs for which there is no significant difference between the original protocol (without buckets) and the protocol that uses buckets. If the performance of these two cases is comparable, one is better off using the version without buckets, since there will be no probability of information being leaked due to bucket overflow.

Another way we could possibly improve the communication and computation complexities of the protocol is by reducing the number of bits used to represent each feature albeit at the cost of increasing the probability of collisions (different features being mapped into the same bit strings). We used 13 bits for representing unigrams and 17 bits for representing unigrams and bigrams. We did not observe any collisions.

[7] https://bitbucket.org/uwtppml

Finally, we note that if the protocol is to be deployed over a wide area network, rather than a local area network, Yao garbled circuits would become a preferable choice for the round intensive parts of our solution (such as in the private feature extraction part).

5 Related work

The interest in privacy-preserving machine learning (PPML) has grown substantially over the last decade. The best-known results in PPML are based on differential privacy (DP), a technique that relies on adding noise to answers, to prevent an adversary from learning information about any particular individual in the dataset from revealed aggregate statistics [30]. While DP in an ML setting aims at protecting the privacy of individuals in the training dataset, our focus is on protecting the privacy of new user data that is classified with proprietary ML models. To this end, we use Secure Multiparty Computation (SMC) [16], a technique in cryptography that has successfully been applied to various ML tasks with structured data (see e.g. [14, 19, 21, 40] and references therein).

To the best of our knowledge there are no existing DP or SMC based solutions for PP feature extraction and classification of unstructured texts. Defenses against authorship attribution attacks that fulfill DP in text classification have been proposed [53]. These methods rely on distortion of term frequency vectors and result in loss of accuracy. In this paper we address a different challenge: we assume that Bob knows Alice, so no authorship obfuscation is needed. Instead, we want to process Alice's text with Bob's classifier, without Bob learning what Alice wrote, and without accuracy loss. To the best of our knowledge, Costantino et al. [15] were the first to propose PP feature extraction from text. In their solution, which is based on homomorphic encryption (HE), Bob learns which of his lexicon's words are present in Alice's tweets, and classification of a single tweet with a model with less than 20 features takes 19 minutes. Our solution does not leak any information about Alice's words to Bob, and classification is done in seconds, even for a model with 500 features.

Below we present existing work that is related to some of the building blocks we use in our PP text classification protocol (see Section 3.1).

Private equality tests have been proposed in the literature based on several different flavors [3]. They can be based on Yao Gates, Homomorphic Encryption, and generic SMC [52]. In our case, we have chosen a simple protocol that depends solely on additions and multiplications over a binary field. While different (and possibly more efficient) comparison protocols could be used instead, they would either require additional computational assumptions or present a marginal improvement in performance for the parameters used here.

Our private feature extraction can be seen as a particular case of private set intersection (PSI). PSI is the problem of securely computing the intersection of two sets without leaking any information except (possibly) the result, such as identifying the intersection of the set of words in a user's text message with the hate speech lexicon used by the classifier. Several paradigms have been proposed to realize PSI functionality, including a Naive hashing solution, Server-aided PSI, and PSI based on oblivious transfer extension. For a complete survey, we refer to Pinkas et al. [45]. In our protocol for PP text classification, we implement private feature extraction by a straightforward application of our equality test protocol. While more efficient protocols could be obtained by using sophisticated hashing techniques, we have decided to stick with our direct solution since it has no probability of failure and works well for the input sizes needed in our problem. For larger input sizes, a more sophisticated protocol would be a better choice [45].

We use two protocols for the secure classification of feature vectors: an existing protocol π_{LR} for *secure classification with LR models* [19]; and a novel *secure AdaBoost classification protocol*. The logistic regression protocol uses solely additions and multiplications over a finite field. The secure AdaBoost classification protocol is an novel optimized protocol that uses solely decision trees of depth one, binary features and a binary output. All these characteristics were used in order to speed up the resulting protocol. The final secure AdaBoost classification protocol uses only two secure inner products and one secure comparison.

Generic protocols for private scoring of machine learning models have been proposed in [8]. The solutions proposed in [8] cannot be used in our setting since they assume that the features' description are publicly known, and thus can be computed locally by Alice and Bob. However, in our case, the features themselves are part of the model and cannot be made public.

Finally, we note that while we implemented our protocols using our own framework for privacy-preserving machine learning [8], any other generic framework for SMC could be also used in principle [47, 22, 41].

6 Conclusion

In this paper we have presented the first provably secure method for privacy-preserving (PP) classification of unstructured text. We have provided an analysis of the correctness and security of solution. As a side result, we also present a novel protocol for binary classification over binary input features with an ensemble of decisions stumps. An implementation of the protocols in Java, run on AWS machines, allowed us to classify text messages securely within seconds. It is important to note that this run time (1) includes both secure feature extraction and secure classification of the extracted feature vector; (2) includes both computation and communication costs, as the parties involved in the protocol were run on separate machines; (3) is two orders of magnitude better than the only other existing solution, which is based on HE. Our results show that in order to make PP text classification practical, one needs to pay close attention not only to the underlying cryptographic protocols but also to the underlying ML algorithms. ML algorithms that would be a clear choice when used in the clear might not be useful at all when transferred to the SMC domain. One has to optimize these ML algorithms having in mind their use within SMC protocols. Our results provide the first evidence that provably secure PP text classification is feasible in practice.

[8] https://bitbucket.org/uwtppml

References

[1] Peter Ray Allison. Tracking terrorists online might invade your privacy. BBC, http://www.bbc.com/future/story/20170808-tracking-terrorists-online-might-invade-your-privacy, 2017.

[2] Tiago A. Almeida, José María G. Hidalgo, and Akebo Yamakami. Contributions to the study of SMS spam filtering: new collection and results. In *Proc. of the 11th ACM Symposium on Document Engineering*, pages 259–262, 2011.

[3] Nuttapong Attrapadung, Goichiro Hanaoka, Shinsaku Kiyomoto, Tomoaki Mimoto, and Jacob CN Schuldt. A taxonomy of secure two-party comparison protocols and efficient constructions. In *15th Annual Conference on Privacy, Security and Trust (PST)*, 2017.

[4] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS 2004*, pages 186–195, 2004.

[5] Paulo S. L. M. Barreto, Bernardo David, Rafael Dowsley, Kirill Morozov, and Anderson C. A. Nascimento. A framework for efficient adaptively secure composable oblivious transfer in the ROM. Cryptology ePrint Archive, Report 2017/993, 2017. `http://eprint.iacr.org/2017/993`.

[6] Valerio Basile, Cristina Bosco, Elisabetta Fersini, Debora Nozza, Viviana Patti, Francisco Rangel, Paolo Rosso, and Manuela Sanguinetti. Semeval-2019 Task 5: Multilingual detection of hate speech against immigrants and women in Twitter. In *Proc. of the 13th International Workshop on Semantic Evaluation (SemEval-2019)*. ACL, 2019.

[7] Donald Beaver. Commodity-based cryptography (extended abstract). In *STOC 1997*, pages 446–455, 1997.

[8] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.

[9] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pages 136–145, 2001.

[10] Ran Canetti and Marc Fischlin. Universally composable commitments. In *Crypto 2001*, pages 19–40, 2001.

[11] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC 2002*, pages 494–503, 2002.

[12] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[13] Michele Ciampi and Claudio Orlandi. Combining private set-intersection with secure two-party computation. In *SCN 2018*, pages 464–482, 2018.

[14] Chris Clifton, Murat Kantarcioglu, Jaideep Vaidya, Xiaodong Lin, and Michael Y. Zhu. Tools for privacy preserving distributed data mining. *ACM SIGKDD Explorations Newsletter*, 4(2):28–34, 2002.

[15] Gianpiero Costantino, Antonio La Marra, Fabio Martinelli, Andrea Saracino, and Mina Sheikhalishahi. Privacy-preserving text mining as a service. In *2017 IEEE Symposium on Computers and Communications (ISCC)*, pages 890–897, 2017.

[16] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

[17] Bernardo David, Rafael Dowsley, Raj Katti, and Anderson CA Nascimento. Efficient unconditionally secure comparison and privacy preserving machine learning classification protocols. In *International Conference on Provable Security*, pages 354–367. Springer, 2015.

[18] Bernardo David, Rafael Dowsley, Jeroen van de Graaf, Davidson Marques, Anderson C. A. Nascimento, and Adriana C. B. Pinto. Unconditionally secure, universally composable privacy preserving linear algebra. *IEEE Transactions on Information Forensics and Security*, 11(1):59–73, 2016.

[19] Martine De Cock, Rafael Dowsley, Caleb Horst, Raj Katti, Anderson Nascimento, Wing-Sea Poon, and Stacey Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE Transactions on Dependable and Secure Computing*, 16(2):217–230, 2019.

[20] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, and Stacey C. Newman. Fast, privacy preserving linear regression over distributed datasets based on pre-distributed data. In *8th ACM Workshop on Artificial Intelligence and Security (AISec)*, pages 3–14, 2015.

[21] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical secure decision tree learning in a teletreatment application. In *International Conference on Financial Cryptography and Data Security*, pages 179–194. Springer, 2014.

[22] Daniel Demmler, Thomas Schneider, and Michael Zohner. Aby-a framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[23] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. pages 164–181.

[24] Rafael Dowsley. *Cryptography Based on Correlated Data: Foundations and Practice*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2016.

[25] Rafael Dowsley, Jörn Müller-Quade, and Anderson C. A. Nascimento. On the possibility of universally composable commitments based on noisy channels. In *SBSEG 2008*, pages 103–114, Gramado, Brazil, September 1–5, 2008.

[26] Rafael Dowsley, Jörn Müller-Quade, and Tobias Nilges. Weakening the isolation assumption of tamper-proof hardware tokens. In *ICITS 2015*, pages 197–213, 2015.

[27] Rafael Dowsley, Jörn Müller-Quade, Akira Otsuka, Goichiro Hanaoka, Hideki Imai, and Anderson C. A. Nascimento. Universally composable and statistically secure verifiable secret sharing scheme based on pre-distributed data. *IEICE Transactions*, 94-A(2):725–734, 2011.

[28] Rafael Dowsley, Jeroen Van De Graaf, Davidson Marques, and Anderson CA Nascimento. A two-party protocol with trusted initializer for computing the inner product. In *International Workshop on Information Security Applications*, pages 337–350. Springer, 2010.

[29] Rafael Dowsley, Jeroen van de Graaf, Jörn Müller-Quade, and Anderson C. A. Nascimento. On the composability of statistically secure bit commitments. *Journal of Internet Technology*, 14(3):509–516, 2013.

[30] Cynthia Dwork. Differential privacy: A survey of results. In *International Conference on Theory and Applications of Models of Computation*, pages 1–19. Springer, 2008.

[31] Brett Hemenway Falk, Daniel Noble, and Rafail Ostrovsky. Private set intersection with linear communication from general assumptions. Cryptology ePrint Archive, Report 2018/238, 2018. `https://eprint.iacr.org/2018/238`.

[32] Golnoosh Farnadi, Geetha Sitaraman, Shanu Sushmita, Fabio Celli, Michal Kosinski, David Stillwell, Sergio Davalos, Marie-Francine Moens, and Martine De Cock. Computational personality recognition in social media. *User Modeling and User-Adapted Interaction*, 26(2-3):109–142, 2016.

[33] Kyle Fritchman, Keerthanaa Saminathan, Rafael Dowsley, Tyler Hughes, Martine De Cock, Anderson Nascimento, and Ankur Teredesai. Privacy-preserving scoring of tree ensembles: A novel framework for AI in healthcare. In *Proc. of 2018 IEEE International Conference on Big Data*, pages 2412–2421, 2018.

[34] Juan A. Garay, Berry Schoenmakers, and José Villegas. Practical and secure solutions for integer comparison. In *PKC 2007*, pages 330–342, 2007.

[35] Tommi Gröndahl, Luca Pajola, Mika Juuti, Mauro Conti, and N. Asokan. All you need is "love": Evading hate-speech detection. In *Proc. of the 11th ACM Workshop on Artificial Intelligence and Security (AISec)*, 2018.

[36] Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In *TCC 2004*, pages 58–76, 2004.

[37] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards. In *MoraviaCrypt 2005*, 2005.

[38] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *Theory of Cryptography*, pages 600–620. Springer, 2013.

[39] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *Eurocrypt 2007*, pages 115–128, 2007.

[40] Selim V Kaya, Thomas B Pedersen, Erkay Savaş, and Yücel Saygıyn. Efficient privacy preserving distributed clustering based on secret sharing. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 280–291. Springer, 2007.

[41] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[42] Bridianne O'Dea, Stephen Wan, Philip J. Batterham, Alison L. Calear, Cecile Paris, and Helen Christensen. Detecting suicidality on Twitter. *Internet Interventions*, 2(2):183–188, 2015.

[43] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *Crypto 2008*, pages 554–571, 2008.

[44] Wouter Penard and Tim van Werkhoven. On the secure hash algorithm family. In *Cryptography in Context*, pages 1–18. 2008.

[45] Benny Pinkas, Thomas Schneider, and Michael Zohner. Scalable private set intersection based on OT extension. *ACM Transactions on Privacy and Security (TOPS)*, 21(2):7, 2018.

[46] Andrew G. Reece, Andrew J. Reagan, Katharina L.M. Lix, Peter Sheridan Dodds, Christopher M. Danforth, and Ellen J. Langer. Forecasting the onset and course of mental illness with Twitter data. *Scientific Reports*, 7(1):13006, 2017.

[47] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 707–721. ACM, 2018.

[48] Ronald L. Rivest. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. Preprint available at http://people.csail.mit.edu/rivest/Rivest- commitment.pdf, 1999.

[49] Mehran Sahami, Susan Dumais, David Heckerman, and Eric Horvitz. A Bayesian approach to filtering junk e-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, volume 62, pages 98–105, 1998.

[50] Rafael Tonicelli, Anderson C. A. Nascimento, Rafael Dowsley, Jörn Müller-Quade, Hideki Imai, Goichiro Hanaoka, and Akira Otsuka. Information-theoretically secure oblivious polynomial evaluation in the commodity-based model. *International Journal of Information Security*, 14(1):73–84, 2015.

[51] Cynthia Van Hee, Gilles Jacobs, Chris Emmery, Bart Desmet, Els Lefever, Ben Verhoeven, Guy De Pauw, Walter Daelemans, and Véronique Hoste. Automatic detection of cyberbullying in social media text. *PloS one*, 13(10):e0203794, 2018.

[52] Thijs Veugen, Frank Blom, Sebastiaan JA de Hoogh, and Zekeriya Erkin. Secure comparison protocols in the semi-honest model. *IEEE Journal of Selected Topics in Signal Processing*, 9(7):1217–1228, 2015.

[53] Benjamin Weggenmann and Florian Kerschbaum. SynTF: Synthetic and differentially private term frequency vectors for privacy-preserving text mining. In *41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 305–314, 2018.

Appendix A Correctness and Security Analysis of Protocols

A.1 Security Model

The gold standard model for proving the security of cryptographic protocols nowadays is the Universal Composability (UC) framework [9] and it is the security model that we use in this work. Protocols that are proven UC-secure enjoy strong securities guarantees and can be arbitrary composed without compromising the security. In short, it is the most adequate model to use when the protocols need to be executed in complex environments such as the Internet, and it additionally allows a modular design of bigger protocols. In this work protocols with two parties, Alice and Bob, are considered and in the following we present an overview of the UC framework for this setting. We refer interested readers to the book of Cramer et al. [16] for more details and the most general definitions.

Apart from the protocol participants, Alice and Bob, there are also an adversary $\mathcal{A}$, an ideal world adversary $\mathcal{S}$ (also known as the simulator) and an environment $\mathcal{Z}$ (which captures everything that happens outside of the instance of the protocol that is being analyzed, and therefore is the one giving the inputs and getting the outputs from the protocol). All these entities are assumed to be interactive Turing machines. The network is assumed to be under adversarial control and therefore $\mathcal{A}$ is the one that delivers the messages between Alice and Bob. In addition to controlling the network scheduling, $\mathcal{A}$ can also corrupt Alice or Bob, in which case he gains the total control over the corrupted party and learn its complete state. For defining the security of the protocol, an ideal functionality $\mathcal{F}$ is defined, which captures the idealized version of what the protocol is supposed to achieve and communicates directly with Alice and Bob to receive the inputs and delivering the outputs of the protocol (in the ideal world, that is all that Alice and Bob do). Then to prove the security of the protocol π, we show that for every possible adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that no environment $\mathcal{Z}$ can distinguish between a real world execution with Alice, Bob and the adversary $\mathcal{A}$ running the protocol π and the ideal world execution with the ideal functionality $\mathcal{F}$, the simulator $\mathcal{S}$ and the dummy version of Alice and Bob that just forward the inputs and outputs between $\mathcal{F}$ and $\mathcal{S}$. Formally:

Definition A.1 ([9]) *A protocol π UC-realizes an ideal functionality $\mathcal{F}$ if, for every possible adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that, for every possible environment $\mathcal{Z}$, the view of the environment $\mathcal{Z}$ in the real world execution with $\mathcal{A}$, Alice and Bob executing the protocol π (with security parameter λ) is computationally indistinguishable from the view of $\mathcal{Z}$ in the ideal world execution with the functionality $\mathcal{F}$, the simulator $\mathcal{S}$ and the dummy Alice and Bob, where the probability distribution is taken over the randomness used by all entities.*

Adversarial Model: We consider honest-but-curious adversaries. Honest-but-curious adversaries follow the protocol instructions correctly, but try to learn additional information. We only consider static adversaries, for which the set of corrupted parties is chosen before the start of the protocol execution and does not change. A version of the UC theorem for the case of honest-but-curious adversaries is given in Theorem 4.20 of Cramer et al. [16].

Setup Assumption: It is a well-known fact that secure two-party computation (and also secure multi-party computation) can only achieve UC-security using a setup assumption [10, 11]. Multiple setup assumptions were used previously to achieve UC-security for secure computation protocols, including: the availability of a common reference string [10, 11, 43], the availability of a public-key infrastructure [4], the random oracle model [36, 5], the existence of noisy channels between the parties [25, 29], and the availability of signature cards [37] or tamper-proof hardware [39, 23, 26]. In this work the commodity-based model [7] is used as the setup assumption. In this model there exists a trusted initializer that pre-distributed correlated randomness to Alice and Bob during a setup phase. This setup phase is run before the protocol execution (and in fact can be performed even before Alice and Bob get to know their inputs), and the trusted initializer does not participate in any other point of the protocol. The commodity-based model was used in many previous works, e.g., [48, 28, 27, 38, 50, 20, 17, 18, 33, 19]. The trusted initializer is modeled by the ideal functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ described in Figure 3.

Simplifications: The simulation strategy in our proofs is in fact very simple: all the computations are performed using secret sharings and all the protocol messages look uniformly random from the point of view of the receiver, with the single exception of the openings of the secret sharings. Nevertheless, the messages that open a secret sharing can be straightforwardly simulated using the outputs of the respective functionalities. In the ideal world, the simulator $\mathcal{S}$ has the leverage of being

Functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$

$\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ is parametrized by an algorithm $\mathcal{D}$. Upon initialization run $(D_A, D_B) \xleftarrow{\$} \mathcal{D}$ and deliver D_A to Alice and D_B to Bob.

Figure 3: The Trusted Initializer Functionality.

Functionality $\mathcal{F}_{\mathsf{DMM}}$

$\mathcal{F}_{\mathsf{DMM}}$ is executed with Alice and Bob is parametrized by the size q of the ring and the dimensions (i, j) and (j, k) of the matrices.

Input: Upon receiving a message from Alice/Bob with her/his shares of $[\![X]\!]_q$ and $[\![Y]\!]_q$, verify if the share of X is in $\mathbb{Z}_q^{i \times j}$ and the share of Y is in $\mathbb{Z}_q^{j \times k}$. If it is not, abort. Otherwise, record the shares, ignore any subsequent message from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both Alice and Bob, reconstruct X and Y from the shares, compute $Z = XY$ and create a secret sharing $[\![Z]\!]_q$. Before the deliver of the output shares, a corrupt party fix its share of the output to any constant value. In both cases the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

Figure 4: The Distributed Matrix Multiplication Functionality.

the one responsible for simulating all the ideal functionalities other than the one whose security is being analyzed (including the trusted initializer functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$), and he can easily use this fact to perform a perfect simulation. For this reason the real and ideal world are indistinguishable for any environment $\mathcal{Z}$ and achieve perfect security.

The messages of the ideal functionalities are formally public delayed outputs, i.e., first the simulator is asked whether it allows the message to be delivered (this is due to the fact that in the real world the adversary controls the scheduling of the network), and the message is only delivered when $\mathcal{S}$ agrees. And formally, every instance has a session identification. We omit those information from descriptions for the sake of readability.

Security of the Building Blocks: The protocol for secure distributed matrix multiplication π_{DMM} UC-realizes the distributed matrix multiplication functionality $\mathcal{F}_{\mathsf{DMM}}$ described in Figure 4 [24, 19]. The protocol for secure comparison π_{DC} UC-realizes the functionality $\mathcal{F}_{\mathsf{DC}}$ described in Figure 5 [34, 19]. The protocol for secure bit-decomposition π_{decomp} UC-realizes the functionality $\mathcal{F}_{\mathsf{decomp}}$ described in Figure 6 [19]. The LR classification protocol π_{LR} UC-realizes the functionality $\mathcal{F}_{\mathsf{LR}}$ described in Figure 7 [19].

The correctness of the equality test protocol π_{EQ} follows from the fact that in the case that $x = y$, then all r_i's will be equal to 1 and therefore $z = \prod_i r_i$ will also be 1. If $x \neq y$, then for at least one value i, we have that $r_i = 0$, and therefore $z = 0$. For the simulation, $\mathcal{S}$ executes an internal copy of $\mathcal{A}$ interacting with an instance of π_{EQ} in which the uncorrupted parties use dummy inputs. Note that all the messages that $\mathcal{A}$ receives look uniformly random to him. Since the share multiplication protocol is substituted by $\mathcal{F}_{\mathsf{DMM}}$ using the UC composition theorem, and $\mathcal{S}$ is the one responsible for simulating $\mathcal{F}_{\mathsf{DMM}}$ in the ideal world, $\mathcal{S}$ can leverage this fact in order to extract the share that any corrupted party have of the value $x_i + y_i$, let the extracted value of the corrupted party be denoted by $v_{i,C}$. The simulator then pick random values $x_{i,C}, y_{i,C} \in \{0, 1\}$ such that $x_{i,C} + y_{i,C} = v_{i,C} \mod 2$ and submit these values to $\mathcal{F}_{\mathsf{EQ}}$ as being the shares of the corrupted party for x_i and y_i (note that the result of $\mathcal{F}_{\mathsf{EQ}}$ only depends on the values of $x_i + y_i \mod 2$). $\mathcal{S}$ is also able to fix the output share of the corrupted party in $\mathcal{F}_{\mathsf{EQ}}$ so that it matches the one in the instance of π_{EQ}. This is a perfect

Functionality $\mathcal{F}_{\mathsf{DC}}$

$\mathcal{F}_{\mathsf{DC}}$ is parametrized by the bit-length ℓ of the values being compared.

Input: Upon receiving a message from Alice/Bob with her/his shares of $[\![x_i]\!]_2$ and $[\![y_i]\!]_2$ for all $i \in \{1, \ldots, \ell\}$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct x and y from the bitwise shares. If $x \geq y$, then create and distribute to Alice and Bob the secret sharing $[\![1]\!]_2$; otherwise the secret sharing $[\![0]\!]_2$. Before the deliver of the output shares, a corrupt party fix its share of the output to any constant value. In both cases the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

Figure 5: The Distributed Comparison Functionality.

Functionality $\mathcal{F}_{\mathsf{decomp}}$

$\mathcal{F}_{\mathsf{decomp}}$ is parametrized by the bit-length ℓ of the value x being converted from an additive secret sharing $[\![x]\!]_q$ in $\mathbb{Z}_q$ to additive bitwise secret sharings $[\![x_i]\!]_2$ in $\mathbb{Z}_2$ such that $x = x_\ell \cdots x_1$.

Input: Upon receiving a message from Alice or Bob with her/his share of $[\![x]\!]_q$, record the share, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of both shares, reconstruct x, compute its bitwise representation $x_\ell \cdots x_1$, and for $i \in \{1, \ldots, \ell\}$ distribute new secret sharings $[\![x_i]\!]_2$ of the bit x_i. Before the output deliver, the corrupt party fix its shares of the outputs to any constant values. The shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraints.

Figure 6: The Bit-Decomposition Functionality.

Functionality $\mathcal{F}_{\mathsf{LR}}$

$\mathcal{F}_{\mathsf{LR}}$ computes the classification according to a logistic regression model with the threshold value set to 0.5. The input feature vector x is secret shared between Alice and Bob.

Input: Upon receiving the weight vector w, the intercept value b and his shares $[\![x_i]\!]_q$ of the elements of x from Bob, or her shares $[\![x_i]\!]_q$ of the elements of x from Alice, store the information, ignore any subsequent message from that party, and inform the other party about the receipt.

Output: Upon getting the inputs from both parties, reconstruct the feature vector x, compute the value $\mathsf{sign}\left(\langle x, w\rangle + b\right)$ and output it to Bob as the class prediction.

Figure 7: The Logistic Regression Classification Functionality.

Functionality $\mathcal{F}_{\mathsf{EQ}}$

$\mathcal{F}_{\mathsf{EQ}}$ is parametrized by the bit-length ℓ of the values being compared.

Input: Upon receiving a message from Alice/Bob with her/his shares of $[\![x_i]\!]_2$ and $[\![y_i]\!]_2$ for all $i \in \{1, \ldots, \ell\}$, record the shares, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct x and y from the bitwise shares. If $x = y$, then create and distribute to Alice and Bob the secret sharing $[\![1]\!]_2$; otherwise the secret sharing $[\![0]\!]_2$. Before the deliver of the output shares, a corrupt party fix its share of the output to any constant value. In both cases the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

Figure 8: The Equality Test Functionality.

simulation strategy, no environment $\mathcal{Z}$ can distinguish the ideal and real worlds and therefore π_{EQ} UC-realizes $\mathcal{F}_{\mathsf{EQ}}$.

The correctness of the secure feature extraction protocol π_{FE} follows directly from the fact that each x_{ij} is equal to 1 if, and only if, $a_j = b_i$, and therefore $x_i = \sum_j x_{ij}$ is equal to 1 if, and only if, b_i is equal to some element of A. In the ideal world, the simulator $\mathcal{S}$ runs internally a copy of $\mathcal{A}$ and an execution of π_{FE} with dummy inputs for the uncorrupted parties. All the messages from the uncorrupted parties look uniformly random from $\mathcal{A}$'s point of view, and therefore the simulation is perfect. $\mathcal{S}$ uses the leverage of being responsible for simulating $\mathcal{F}_{\mathsf{EQ}}$ (π_{EQ} is substituted by $\mathcal{F}_{\mathsf{EQ}}$ using the UC composition theorem) in order to extract the inputs of any corrupted party and forward it to $\mathcal{F}_{\mathsf{FE}}$. No environment $\mathcal{Z}$ can distinguish the ideal world from the real one, and thus π_{FE} UC-realizes $\mathcal{F}_{\mathsf{FE}}$.

In the case of the conversion protocol π_{2toQ} the correctness of the protocol execution follows straightforwardly: since $x = x_a + x_B \mod 2$, then $z = x_A + x_B - 2x_A x_B$ is such that $z = x$ for all possible values $x_A, x_B \in \{0, 1\}$. As for the security, the simulator $\mathcal{S}$ runs internally a copy of the adversary $\mathcal{A}$ and simulates to him an execution of the protocol π_{2toQ} using dummy inputs for the uncorrupted parties. As all the messages from the uncorrupted parties look uniformly random from the adversary point of view, and so the simulation is perfect. The simulator can use the fact that it is the one simulating the multiplication functionality $\mathcal{F}_{\mathsf{DMM}}$ (the secret sharing multiplication is substituted by $\mathcal{F}_{\mathsf{DMM}}$ using the UC composition theorem) in order to extract the share of any corrupted party and fix the input to/output from $\mathcal{F}_{\mathsf{2toQ}}$ appropriately, so that no environment $\mathcal{Z}$ can distinguish the real and ideal worlds. Hence π_{2toQ} UC-realizes $\mathcal{F}_{\mathsf{2toQ}}$.

The AdaBoost classification protocol π_{AB} is trivially correct for the case of binary features and output class, and decision stumps. In the simulation, $\mathcal{S}$ runs an internal copy of $\mathcal{A}$ interacting with a simulated instance of π_{AB} that uses dummy inputs for the uncorrupted parties. π_{IP} is substituted by $\mathcal{F}_{\mathsf{DMM}}$ using the UC composition theorem. $\mathcal{S}$ uses the leverage of simulating $\mathcal{F}_{\mathsf{DMM}}$ in order to extract the shares of the feature vector belonging to a corrupted party, as well as the weighted probability vectors y and z if Bob is corrupted. $\mathcal{S}$ can then give these extracted inputs to $\mathcal{F}_{\mathsf{AB}}$. No environment can distinguish the real and ideal worlds since the simulation is perfect, and thus π_{AB} UC-realizes $\mathcal{F}_{\mathsf{AB}}$.

Security of the Privacy-Preserving Text Classification Solutions:

The protocol $\pi_{\mathsf{TC-LR}}$ simply executes sequentially the protocols π_{FE}, π_{2toQ} and π_{LR}. Given that these protocols UC-realize $\mathcal{F}_{\mathsf{FE}}$, $\mathcal{F}_{\mathsf{2toQ}}$ and $\mathcal{F}_{\mathsf{LR}}$, respectively, they can be substituted by the functionalities using the UC composition theorem. Note that the sequential composition of those functionalities trivially perform the same computation as $\mathcal{F}_{\mathsf{TC-LR}}$, and no information other than the output of the classification is revealed (all the intermediate values are kept as secret sharings). In the ideal world $\mathcal{S}$ simulates an internal copy of the adversary $\mathcal{A}$ running $\pi_{\mathsf{TC-LR}}$ and using dummy inputs for the uncorrupted parties. The simulator $\mathcal{S}$ can easily extract all the information (from the corrupted parties) that it needs to provide to $\mathcal{F}_{\mathsf{TC-LR}}$ by using the leverage of being responsible for simulating

Functionality $\mathcal{F}_{\mathsf{FE}}$

$\mathcal{F}_{\mathsf{FE}}$ is parametrized by the sizes m of Alice's set and n of Bob's set, and the bit-length ℓ of the elements.

Input: Upon receiving a message from Alice with her set $A = \{a_1, a_2, \ldots, a_m\}$ or from Bob with his set $B = \{b_1, b_2, \ldots, b_n\}$, record the set, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, define the binary feature vector x of length n by setting each element x_i to 1 if $b_i \in A$, and to 0 otherwise. Then create and distribute to Alice and Bob the secret sharings $[\![x_i]\!]_2$. Before the deliver of the output shares, a corrupt party fix its share of the output to any constant value. In both cases the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

Figure 9: The Secure Feature Extraction Functionality.

Functionality $\mathcal{F}_{\mathsf{2toQ}}$

$\mathcal{F}_{\mathsf{2toQ}}$ is parametrized by the size of the field q.

Input: Upon receiving a message from Alice/Bob with her/his share of $[\![x]\!]_2$, record the share, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, reconstruct x, then create and distribute to Alice and Bob the secret sharing $[\![x]\!]_q$. Before the deliver of the output shares, a corrupt party fix its share of the output to any constant value. In both cases the shares of the uncorrupted parties are then created by picking uniformly random values subject to the correctness constraint.

Figure 10: The Secret Sharing Conversion Functionality.

Functionality $\mathcal{F}_{\mathsf{AB}}$

$\mathcal{F}_{\mathsf{AB}}$ computes the classification according to AdaBoost with multiple decision stumps. All the features are binary and the output class is also binary. The input feature vector x is secret shared between Alice and Bob. The model specified by Bob can be expressed in a simplified way by two weighted probability vectors $y = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1})$ and $z = (z_{1,0}, z_{1,1}, \ldots, z_{n,0}, z_{n,1})$. For the i-th decision stump: $y_{i,k}$ is the weighted probability (i.e., a probability multiplied by the weight of the i-th decision stump) that the model assigns to the output class being 0 if $x_i = k$, and $z_{i,k}$ is defined similarly for the output class 1.

Input: Upon receiving the vectors y and z and his shares $[\![x_i]\!]_q$ of the elements of the feature vector x from Bob, or her shares $[\![x_i]\!]_q$ of the elements of x from Alice, store the information, ignore any subsequent message from that party, and inform the other party about the receipt.

Output: Upon getting the inputs from both parties, reconstruct the feature vector x and let $w = (1 - x_1, x_1, 1 - x_2, x_2, \ldots, 1 - x_n, x_n)$. If $\langle w, z \rangle \geq \langle w, y \rangle$, output the class prediction 1 to Bob; otherwise output 0.

Figure 11: The AdaBoost Classification Functionality.

Functionality $\mathcal{F}_{\mathsf{TC-LR}}$

$\mathcal{F}_{\mathsf{TC-LR}}$ computes the privacy-preserving text classification according to a logistic regression model with the threshold value set to 0.5. It is parametrized by the sizes m of Alice's set and n of Bob's set, and the bit-length ℓ of the elements.

Input: Upon receiving a message from Alice with her set $A = \{a_1, a_2, \ldots, a_m\}$ or from Bob with his set $B = \{b_1, b_2, \ldots, b_n\}$, the weight vector w and the intercept value b, record the values, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon getting the inputs from both parties, define the feature vector x of length n as follows: $x_i = 1$ if $b_i \in A$; and $x_i = 0$ otherwise. Compute the value sign $(\langle x, w\rangle + b)$ and output it to Bob as the class prediction.

Figure 12: The Functionality for Privacy-Preserving Text Classification with Logistic Regression.

Functionality $\mathcal{F}_{\mathsf{TC-AB}}$

$\mathcal{F}_{\mathsf{TC-AB}}$ computes the privacy-preserving text classification according to AdaBoost with multiple decision stumps. It is parametrized by the sizes m of Alice's set and n of Bob's set, and the bit-length ℓ of the elements. All the features are binary and the output class is also binary. The model specified by Bob can be expressed in a simplified way by two weighted probability vectors $y = (y_{1,0}, y_{1,1}, \ldots, y_{n,0}, y_{n,1})$ and $z = (z_{1,0}, z_{1,1}, \ldots, z_{n,0}, z_{n,1})$. For the i-th decision stump: $y_{i,k}$ is the weighted probability (i.e., a probability multiplied by the weight of the i-th decision stump) that the model assigns to the output class being 0 if the feature $x_i = k$, and $z_{i,k}$ is defined similarly for the output class 1.

Input: Upon receiving a message from Alice with her set $A = \{a_1, a_2, \ldots, a_m\}$ or from Bob with his set $B = \{b_1, b_2, \ldots, b_n\}$, y and z, record the values, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon getting the inputs from both parties, define the feature vector x of length n as follows: $x_i = 1$ if $b_i \in A$; and $x_i = 0$ otherwise. Let $w = (1-x_1, x_1, 1-x_2, x_2, \ldots, 1-x_n, x_n)$. If $\langle w, z\rangle \geq \langle w, y\rangle$, output the class prediction 1 to Bob; otherwise output 0.

Figure 13: The Functionality for Privacy-Preserving Text Classification with Adaboost.

$\mathcal{F}_{\mathsf{FE}}$, $\mathcal{F}_{\mathsf{2toQ}}$ and $\mathcal{F}_{\mathsf{LR}}$ in the ideal world. Therefore no environment $\mathcal{Z}$ can distinguish the real world from the ideal world, and $\pi_{\mathsf{TC-LR}}$ UC-realizes $\mathcal{F}_{\mathsf{TC-LR}}$.

Similarly, the protocol $\pi_{\mathsf{TC-AB}}$ just runs sequentially the protocols π_{FE}, π_{2toQ} and π_{AB}, that can be substituted by $\mathcal{F}_{\mathsf{FE}}$, $\mathcal{F}_{\mathsf{2toQ}}$ and $\mathcal{F}_{\mathsf{AB}}$ using the UC composition theorem. The result of the computation is trivially the same as in $\mathcal{F}_{\mathsf{TC-AB}}$, and no additional information is revealed. $\mathcal{S}$ runs internally a copy of $\mathcal{A}$ interacting with a simulated instance of $\pi_{\mathsf{TC-AB}}$ (using dummy inputs for the uncorrupted parties) and can easily extract from the corrupted parties all the information that it must provide to $\mathcal{F}_{\mathsf{TC-AB}}$ by using the leverage of being responsible for simulating $\mathcal{F}_{\mathsf{FE}}$, $\mathcal{F}_{\mathsf{2toQ}}$ and $\mathcal{F}_{\mathsf{AB}}$ in the ideal world. No environment $\mathcal{Z}$ can distinguish the real and ideal worlds, and therefore $\pi_{\mathsf{TC-AB}}$ UC-realizes $\mathcal{F}_{\mathsf{TC-AB}}$.

Chapter 4

CONCLUSION

4.1 Individual Contributions

For the work on logistic regression training, I was involved in the initial phase of participating in the iDASH competition. Specifically, I was involved in the testing of logistic regression as a possible model that we would want to use to train in a privacy-preserving way, and I implemented logistic regression training in-the-clear for our initial testing phases. I also helped to break down the training algorithm into corresponding secure protocols to be implemented or created. When we needed to compare our work to an existing fast implementation (SecureML), I worked on getting a running version of their code to train on our data, ran experiments using their code, and worked to understand their protocols/implementation to know where we could improve from their work.

For the work on privacy-preserving text classification, I was involved in the implementation of a few secure protocols. Specifically, using the Lynx framework in Java, I implemented the secure equality test protocol π_{EQ}, the secure logistic regression classification protocol π_{LR}, and the secure AdaBoost classification protocol π_{AB}.

4.2 Future Work

While our implementation of the secure logistic regression training is fast, there are still improvements that can be made to make it even more efficient. While our secure batch bit decomposition protocol is already an improvement over using the secure bit decomposition protocol multiple times (linear time), there is a secure bit decomposition protocol that runs in logarithmic time that could be incorporated instead. To improve the runtime and communication cost of the multiplication triples, we could generate multiplication triples using a

pseudorandom number generator and a specified seed. This might be a route to incorporate into our future work. For our implementation of the private text classification, it may be worth spending time to migrate it over to a faster programming language, such as the RUST programming language. RUST was used for the logistic regression training, which proved to be faster than the java implementation.

Chapter 5

APPENDIX A

Privacy Preserving Classification of High Dimensional Gene Expression Data

Martine De Cock[1], Rafael Dowsley[2], Anderson Nascimento[1], Davis Railsback[1], Jianwei Shen[1], and Ariel Todoki[1]
mdecock@uw.edu, rafael@dowsley.net, andclay@uw.edu, drail@uw.edu, sjwjames@uw.edu, atodoki@uw.edu
[1]School of Engineering and Technology, University of Washington Tacoma
[2]Department of Computer Science, Bar-Ilan University

Introduction

Machine learning has many applications in the biomedical field, allowing for medical diagnoses and personalized medicine.

Biomedical datasets have some challenges:

High dimensional data
- Large number of features
- Small sample size

Data is often split between parties
- Privacy regulations and concerns restrict sharing

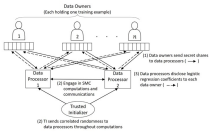
Winner of iDASH2019 – Track IV
Competition on Secure Genome Analysis

Track IV: Secure Collaborative Training of Machine Learning Model [1]

Train a binary classifier on high dimensional gene expression data which is shared between different **data owners**.

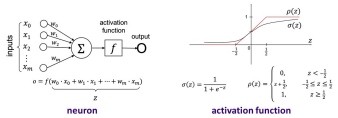
Requirements:
- Multiple data owners who send encrypted or secret shared data to **data processors**
- Data processors do not learn any of the data
- Solution follows the security standard for Secure Multiparty Computation (SMC)
- Secure in the **semi-honest** cryptographic model of security

Overview of Secure Logistic Regression Training:



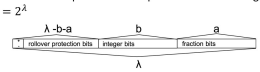
Method: Secure Logistic Regression

Algorithm for secure evaluation of activation function



SMC Based Logistic Regression (LR) training:

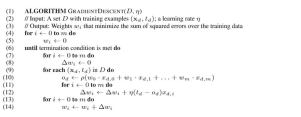
1. Data owners map real-valued inputs to values of a ring $\mathbb{Z}_q$ with $q = 2^\lambda$



2. Data owners split each training example into **additive secret shares** and send secret sharings $[[\vec{x}_d]]_q$, $[[\vec{t}_d]]_q$ to data processors.
3. Data processors run a secure version of the gradient descent algorithm to compute a secret sharing $[[\vec{w}]]_q$ of the neuron's weight vector.

All computations over shares by the data processors happen locally, except for:
- the dot product (**line 10**)
- the activation function (**line 10**)
- and the multiplication (**line 12**)

We use the Trusted Initializer (TI) to distribute randomness in the form of **multiplication triples**.



New protocol for computation of activation function $\rho(z)$:

- Works by leveraging the fact that comparing a secret-shared value over a power of 2 ring with a public power of 2 is computationally easier than a typical SMC comparison
- The most significant bit is extracted from z, and $|z|$ is calculated with secure conditional assignment
- Next, $|z|$ is decomposed and used to extract (1) the integer component + the a-th most significant bit, (2) the fractional component without the a-th most significant bit
- The bitwise OR over all values in (1) is taken – this is logically equivalent to (3) $|z| \geq ½$
- The most significant bit, (2), and (3) are sufficient to determine the output region of $\rho(z)$

What makes our approach fast? [3]

1. A novel protocol implementing the activation function that:
   - does not require transforming to Yao's garbled circuits
   - does not require a costly comparison protocol
2. A very fast implementation based on state-of-the-art cryptographic engineering techniques that has perfect security against semi-honest adversaries that takes advantage of low-level software and memory features to streamline the process of computing massive amounts of multiplication protocols
3. A new approach to bit decomposing (that is, switching from shares over $\mathbb{Z}_{2^n}$ to n shares over $\mathbb{Z}_2$) for large quantities of numbers that does not increase the number of communications

Results

Two data sets with breast cancer tissue samples:

	BC-TCGA	GSE2034
# Features (genes)	17,814	12,634
# Negative Samples	48	83
# Positive Samples	422	142
5-Fold Cross Validation Accuracy	99.58%	64.82%
Training Runtime on 80% of data	2.15(min)	15.04 (min)
Number of Iterations[1]	10	223
Learning Rate		0.001

Implementation was done in RUST [3]
Evaluation is done in AWS:

Hardware:
c5.9xlarge, 36 vCPUs, 72 GiB Memory
Gigabit Ethernet

Configurations:
Integer precision b= 15
Fractional precision a = 12
Ring size lambda λ = 64

(First place in iDASH2019 Track IV competition)

[1] We do not use early stopping as that leaks information. We use a fixed number of iterations instead.

Conclusion

- We receive expected accuracy for both datasets
- Training time was well within the competition requirement of 24 hours
- Data processors learn nothing about the values in the dataset due to data owners sending secret shares of their data
- Our protocol works when data is horizontally partitioned, and when data is vertically partitioned

References

[1] http://www.humangenomeprivacy.org/2019
[2] Payman Mohassel and Yupeng Zhang. SecureML: A system for scalable privacy-preserving machine learning. In 2017 IEEE Symposium on Security and Privacy (SP), pages 19–38.
[3] https://bitbucket.org/uwtppml/idash2019-rust-martine/src/master/

SCHOOL OF ENGINEERING & TECHNOLOGY
UNIVERSITY of WASHINGTON | TACOMA

Bar-Ilan University