TECHNICAL ADVANCE ARTICLE

High Performance Logistic Regression for Privacy-Preserving Genome Analysis

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

Abstract

Background: In biomedical applications, valuable data is often split between owners 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 the cryptographic technique named secure Multi-Party Computation, the price paid for keeping the data of the owners 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 data sets. 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 that distributes correlated randomness to the two computing parties. We use a gradient descent based algorithm for training a logistic regression model, and we break down the algorithm into corresponding cryptographic protocols. Our main contributions are a new protocol for computing the activation function that requires neither secure comparison protocols nor Yao's garbled circuits, and a series of cryptographic engineering optimizations to improve the performance.

Results: For our largest gene expression data set, we train a model that requires over 7 billion secure multiplications; the training completes in about 26.90 seconds in a local area network. The implementation in this work is a further optimized version of the implementation with which we won first place in Track 4 of the iDASH 2019 secure genome analysis competition.

Conclusions: In this paper, we present a secure logistic regression training protocol and its implementation, with a new subprotocol to securely compute the activation function. To the best of our knowledge, we present the fastest existing secure Multi-Party Computation implementation for training logistic regression models on high dimensional genome data distributed across a local area network.

Keywords: Logistic regression; Gradient descent; Machine Learning; Secure Multi-Party Computation; Gene expression data

Background

Introduction

Machine Learning (ML) has many applications in the biomedical domain, such as medical diagnosis and personalized medicine. Biomedical data sets 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, valuable 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, perform 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

[*] 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

by different data owners, while keeping the training data private. This work is directly inspired by Track 4 of the iDASH 2019 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 and advancing new techniques to solve real-world problems in handling genomic data. In the 2019 edition there were a total of four different tracks, where Track 4 invited participants to design MPC solutions for collaborative training of ML models originating from multiple data owners. One of the Track 4 competition data sets 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 implied 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 1. 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 data set, 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 features 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 data set are: (i) a new protocol for securely computing the activation function that avoids the use of full-fledged secure comparison protocols; (ii) a novel method for bit decomposing secret shared integers and bundling their 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 (LANs). 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 iDASH 2019 competition. For our largest data set, we train a model that requires over 7 billion secure multiplications and the training completes in about 26.9 seconds in a LAN.

This paper significantly expands over a preliminary version of this result [2], presented at a workshop without formal proceedings. In this version we have a formal description of all protocols, security proofs and improved running times.

We first discuss below our work as compared to others. In the Section **Methods**, we present preliminary information on MPC, describe the secure subprotocols that are building blocks for our secure LR training protocol, and finally describe the protocol itself. In the Section **Results** we describe details of our implementation and runtime results for the overall protocol and microbenchmarks for our secure activation function protocol. We experimentally compare our solution with the state-of-the-art SecureML approach [3], demonstrating substantial runtime improvements. In the Section **Discussion**, we note possible future work to improve and extend our results, and finally in the Section **Conclusions** we present our summary remarks.

Related Work

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

One scenario that was considered in previous works [4, 5, 6] 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 an MPC protocol among themselves to train the model, or delegate the computation to a set of data processors that run a MPC protocol. It is the latter setting that we follow in this paper.

Existing MPC approaches to secure LR differ in the numerical optimization algorithms used for LR training and in the cryptographic primitives leveraged [7, 3, 8, 9]. The SPARK protocol [7] 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

[1] http://www.humangenomeprivacy.org/2019/competition-tasks.html, accessed on Jan 19, 2020

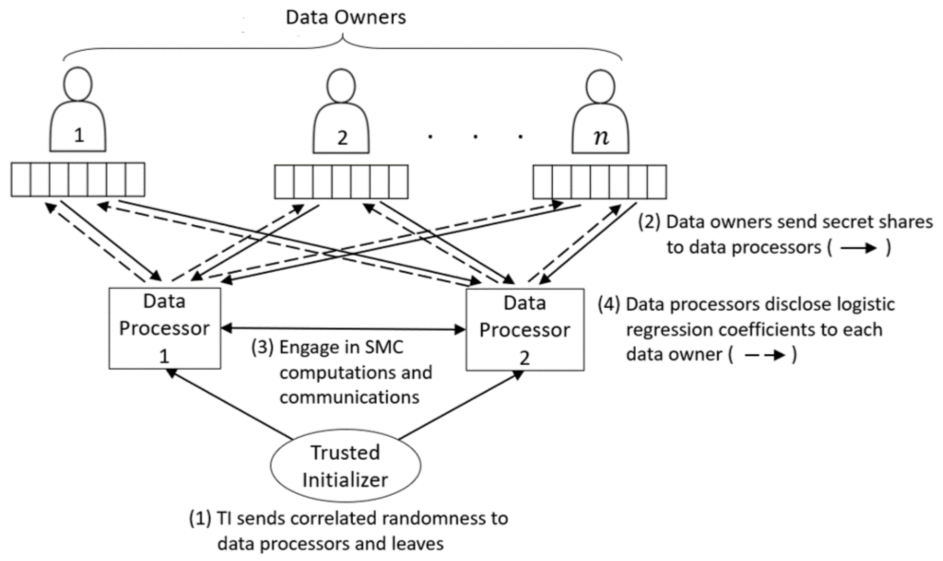

Data Owners
1
2
n
(2) Data owners send secret shares to data processors (⟶)
Data Processor 1
Data Processor 2
(4) Data processors disclose logistic regression coefficients to each data owner (- -→)
(3) Engage in SMC computations and communications
Trusted Initializer
(1) TI sends correlated randomness to data processors and leaves

Figure 1 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.

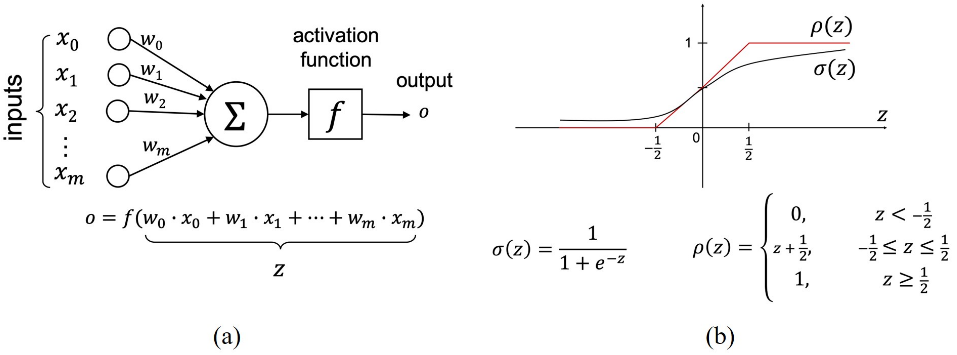

inputs
x_0
x_1
x_2
x_m
w_0
w_1
w_2
w_m
Σ
activation function
f
output
o
$o = f(\underbrace{w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_m \cdot x_m}_{z})$
(a)
$\rho(z)$
$\sigma(z)$
1
z
$-\frac{1}{2}$
0
$\frac{1}{2}$
$\sigma(z) = \frac{1}{1+e^{-z}}$
$\rho(z) = \begin{cases} 0, & z < -\frac{1}{2} \\ z + \frac{1}{2}, & -\frac{1}{2} \leq z \leq \frac{1}{2} \\ 1, & z \geq \frac{1}{2} \end{cases}$
(b)

Figure 2 Architecture (a) Neuron; (b) Approximation of sigmoid activation function σ by clipped ReLu ρ

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 [8] 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 [9] uses Yao's garbled circuits and Paillier encryption; their protocol 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 [7], would not align with the iDASH 2019 competition requirements. We also point out a protocol secure against active adversaries from SecureNN [10] 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 [3], which was the fastest protocol for privately training LR models based on secure MPC prior to our work. SecureML separates the data owners from the data processors, and uses mini-batch gradient descent. The main novelty points of SecureML are a clipped ReLu activation function, 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 data set 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 [7, 8, 9]. Like our solution, the SecureML protocol is split into an offline and online phase (the offline phase can be executed before the inputs are known and is responsible for generating multiplication triples). The SecureML solution is based on two servers, while our solution is based on three servers, namely a party who pre-computes so-called multiplication triples in the offline stage, and two parties who actively compute the final result. If we exclude the preprocessing/off-line stage from SecureML and exclude the pre-distribution of triples in our solution, we are left with protocols that work in exactly the same setting. We compare the runtime of both solutions in the Section **Results**, showing that our implementation is substantially faster.

A preliminary version of this work appeared in a workshop without formal proceedings [2]. This paper is a substantially longer and detailed description that includes security proofs, detailed comparison with the state-of-the-art, and improved running times.

Methods

Logistic Regression

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 2(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 $o_d = f(w_0 \cdot x_{d,0} + w_1 \cdot x_{d,1} + \cdots + w_n \cdot x_{d,n})$, which is interpreted as the probability that the class label is 1. 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 MPC, we approximate it with the activation function ρ from [3], which is shown in Figure 2(b).

Algorithm 1: Full Gradient Descent

```
Input  : A set $D$ with training examples $(\boldsymbol{x}_d, t_d)$;
         a learning rate $\eta$
Output: Weights $w_i$ that minimize the sum of
         squared errors over the training data
1  for $i \leftarrow 0$ to $m$ do
2      $w_i \leftarrow 0$
3  until termination condition is met do
4      for $i \leftarrow 0$ to $m$ do
5          $\Delta w_i \leftarrow 0$
6      for each $(\boldsymbol{x}_d, t_d)$ in $D$ do
7          $o_d \leftarrow \rho(w_0 \cdot x_{d,0} + w_1 \cdot x_{d,1} + \ldots + w_m \cdot x_{d,m})$
8          for $i \leftarrow 0$ to $m$ do
9              $\Delta w_i \leftarrow \Delta w_i + \eta(t_d - o_d)x_{d,i}$
10     for $i \leftarrow 0$ to $m$ do
11         $w_i \leftarrow w_i + \Delta w_i$
12 return $w_0, \ldots, w_m$
```

For training, we use the full gradient descent based algorithm shown in Algorithm 1 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 [8]. Instead, we use a fixed number of iterations during training.

Our scenario

In the scenario considered in this work the data is not held by a single party that performs all the computation, but distributed by the data owners to the data processors in such way that each data processor does not have any information about the data in the clear. Nevertheless, the data processors would still like to compute a LR model without leaking any other information about the data used for the training. To achieve this goal, we will use techniques from MPC.

Our setup is illustrated in Figure 1. We have multiple data owners who each hold disjoint parts of the data that is going to be used for the training. This is the most general approach and covers the cases in which the data is horizontally partitioned (i.e. for each training sample $d = (\boldsymbol{x}_d, t_d)$, all the data for d is held by one of the data owners), vertically partitioned (for each feature, the values of that feature for all training samples are held by one of the data owners), and even arbitrary partitions. There are two data processors who collaborate to train a LR model using secure MPC protocols, and a trusted initializer (TI) that predistributes correlated randomness to the data processors in order to make the MPC computation more efficient. The TI is not involved in any other part of the

[2] This is a technique that uses a metric, such as the accuracy on a held-out validation data set, to check when a model starts to overfit and will then stop training at that point.

execution, and does not learn any data from the data owners or data processors.

We next present the security model that is used and several secure building blocks, so that afterwards we can combine them in order to obtain a secure LR training protocol.

Security Model

The security model in which we analyze our protocol is the Universal Composability (UC) framework [11] 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. [12] 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 necessity 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 (the data processors from Figure 1, 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 single execution of the protocol that is under consideration). 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 of the functionality that the protocol is supposed to perform is defined. The ideal functionality $\mathcal{F}$ 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 executing in the real world 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 in the real world 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 like the vast majority of the privacy-preserving machine learning protocols in the literature considers honest-but-curious, static adversaries. In more detail, the adversary chooses the party that he wants to corrupt before the protocol execution and he also follows the protocol instructions (but tries to learn additional information). We consider the trusted initializer model, in which a trusted initializer functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ pre-distributes correlated randomness to Alice and Bob.[3] A trusted initializer has been often used to enable highly efficient solutions both in the context of privacy-preserving machine learning [23, 24, 25, 26, 27] as well as in other applications, e.g., [28, 29, 30, 31, 32, 33].

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 $(D_A, D_B) \overset{\$}{\leftarrow} \mathcal{D}$, and deliver D_A to Alice and D_B to 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 a secret shared value 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 Multi-Party Computation

Our MPC solution is based on additive secret sharing over a ring $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$. 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 in $\mathbb{Z}_q$ with the constraint that $x_A + x_B = x$

[3] Using a setup assumption, like the trusted initializer, in the MPC protocol is a necessity in order to get UC-security [13, 14]. Other possible setup assumption to achieve UC-security include: a common reference string [13, 14, 15], the availability of a public-key infrastructure [16], the random oracle model [17, 18], the existence of noisy channels between the parties [19, 20], and the availability of tamper-proof hardware [21, 22].

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. The secret shared 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 share 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 $[\![z]\!]_q \leftarrow [\![x]\!]_q + c$.

The secure multiplication of secret shared values (i.e., $z = xy$) cannot be done locally and involves communication between Alice and Bob. To obtain an efficient secure multiplication solution, we use the multiplication triples technique that was originally proposed by Beaver [34]. 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 [26, 35] and denote by π_{DM} the protocol for the special case of multiplication of scalars and π_{IP} for the inner product. As shown in [26] the protocol π_{DMM} (described in Protocol 2) UC-realizes the distributed matrix multiplication functionality $\mathcal{F}_{\mathsf{DMM}}$ in the trusted initializer model.

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

$\mathcal{F}_{\mathsf{DMM}}$ runs with Alice and Bob and is parametrized by the size q of the ring $\mathbb{Z}_q$ 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 fixes 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.

Protocol 2: Secure Distributed Matrix Multiplication Protocol π_{DMM}

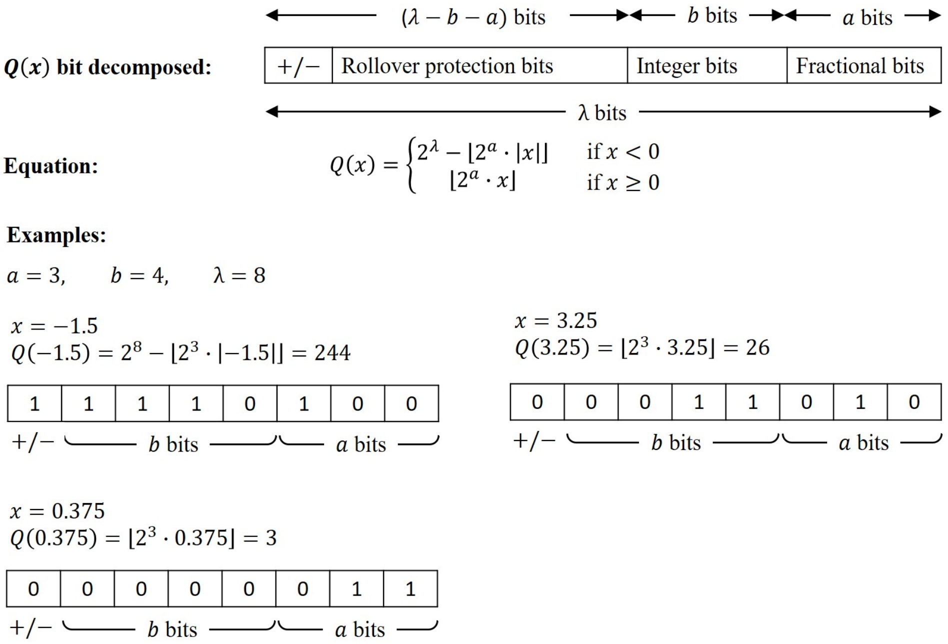
```
Input  : $[\![X]\!]_q$, $[\![Y]\!]_q$
Output: $[\![Z]\!]_q$ such that $Z = XY$
The protocol is parametrized by the size $q$ of the ring $\mathbb{Z}_q$ and the dimensions $(i, j)$ and $(j, k)$ of the matrices. The trusted initializer chooses uniformly random $U$ and $V$ in $\mathbb{Z}_q^{i\times j}$ 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.
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 $E$.
Alice and Bob locally compute $[\![Z]\!]_q \leftarrow [\![W]\!]_q + E[\![U]\!]_q + D[\![V]\!]_q + DE$.
return $[\![Z]\!]_q$
```

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 3, 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



$(\lambda - b - a)$ bits
b bits
a bits
$Q(x)$ bit decomposed:
+/−
Rollover protection bits
Integer bits
Fractional bits
λ bits
Equation:
$Q(x) = \begin{cases} 2^\lambda - \lfloor 2^a \cdot |x| \rfloor & \text{if } x < 0 \\ \lfloor 2^a \cdot x \rfloor & \text{if } x \geq 0 \end{cases}$
Examples:
$a = 3, \quad b = 4, \quad \lambda = 8$
$x = -1.5$
$Q(-1.5) = 2^8 - \lfloor 2^3 \cdot |-1.5| \rfloor = 244$
1 1 1 1 0 1 0 0
+/− b bits a bits
$x = 3.25$
$Q(3.25) = \lfloor 2^3 \cdot 3.25 \rfloor = 26$
0 0 0 1 1 0 1 0
+/− b bits a bits
$x = 0.375$
$Q(0.375) = \lfloor 2^3 \cdot 0.375 \rfloor = 3$
0 0 0 0 0 0 1 1
+/− b bits a bits

Figure 3 Fixed-Point Representation Register map of fixed-point representation of numbers shared over $\mathbb{Z}_{2^\lambda}$ with examples.

represent the largest number x that can be produced during the LR protocol, and therefore λ should be chosen to be at least $2(a + b)$ (see Truncation). 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 3.

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, for $x, y > 0$, 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 local truncation protocol for fixed point representations of real numbers proposed in [3] that we will refer to as π_{trunc}. It does not involve any messages between the two parties, each party simply performs an operation on its own local share. This protocol almost 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 (inner product) of the current weights vector with a training example in line 7 of Algorithm 1, and (2) when the weight differentials (Δw_i) are adjusted in line 9 of Algorithm 1. 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 is 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 between the secure version and the plaintext version.

Conversion of Sharings

For efficiency reasons, in some of the steps for securely computing the activation function we use secret sharings over $\mathbb{Z}_2$, while in others we use secret sharings over $\mathbb{Z}_{2^\lambda}$. Therefore we need to be able to convert between the two types of secret sharings.

We use the two-party protocol from [26] 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 the two additive shares held by the parties and an "XOR-sharing" of that sum is the carry vector. As proven in [26], the bit-decomposition protocol π_{decomp} (described in Protocol 3) UC-realizes the bit-decomposition functionality $\mathcal{F}_{\mathsf{decomp}}$.

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

$\mathcal{F}_{\mathsf{decomp}}$ 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 $[\![x]\!]_{2^\lambda}$, 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.

In our implementation we use a highly parallelized and optimized version of the bit-decomposition protocol π_{decomp} in order to improve the communication efficiency of the overall solution. The optimizations are described in the Appendix.

Protocol 3: 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 $x$.
All distributed multiplications are over $\mathbb{Z}_2$ and the
  required correlated randomness is pre-distributed
  by the trusted initializer.
Let $a$ denote Alice's share of $x$, which corresponds
  to a bit string $a_\lambda \ldots a_1$. Similarly, let $b$ denote
  Bob's share of $x$, which corresponds to a 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 $(0, b_i)$.
Compute $[\![c_1]\!]_2 \leftarrow [\![a_1]\!]_2 [\![b_1]\!]_2$ and set
  $[\![x_1]\!]_2 \leftarrow [\![y_1]\!]_2$.
for $i \leftarrow 2$ to $\lambda$ do
    Compute $[\![d_i]\!]_2 \leftarrow [\![a_i]\!]_2 [\![b_i]\!]_2 + 1$.
    $[\![e_i]\!]_2 \leftarrow [\![y_i]\!]_2 [\![c_{i-1}]\!]_2 + 1$
    $[\![c_i]\!]_2 \leftarrow [\![e_i]\!]_2 [\![d_i]\!]_2 + 1$
    $[\![x_i]\!]_2 \leftarrow [\![y_i]\!]_2 + [\![c_{i-1}]\!]_2$
end
return $[\![x_i]\!]_2$ for $i \in \{1, \ldots, \lambda\}$
```

The opposite of a secure bit-decomposition is converting from bit sharing to an additive sharing over a larger ring. In our secure activation function protocol, we require securely converting a bit sharing to an additive sharing in 2^λ. This is done using the protocol $\pi_{\mathsf{2to2^\lambda}}$ from [27] (described in Protocol 4) that UC-realizes the secret sharing conversion functionality $\mathcal{F}_{\mathsf{2to2^\lambda}}$.

Functionality $\mathcal{F}_{\mathsf{2to2^\lambda}}$

$\mathcal{F}_{\mathsf{2to2^\lambda}}$ is parametrized by the bit-length λ of the ring in which the output is shared.

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

Protocol 4: Secure Secret Sharing Conversion Protocol $\pi_{\mathsf{2to2^\lambda}}$

```
Input  : $[\![x]\!]_2$
Output: $[\![x]\!]_{2^\lambda}$
All distributed multiplications are over $\mathbb{Z}_{2^\lambda}$ and
  the required correlated randomness is
  pre-distributed by the trusted initializer.
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 $[\![x_A]\!]_{2^\lambda}$ by picking
  uniformly random shares that sum to $x_A$ and
  delivers Bob's share to him, and Bob proceeds
  similarly to create $[\![x_B]\!]_{2^\lambda}$.
$[\![y]\!]_{2^\lambda} \leftarrow [\![x_A]\!]_{2^\lambda} [\![x_B]\!]_{2^\lambda}$
$[\![z]\!]_{2^\lambda} \leftarrow [\![x_A]\!]_{2^\lambda} + [\![x_B]\!]_{2^\lambda} - 2[\![y]\!]_{2^\lambda}$
return $[\![z]\!]_{2^\lambda}$
```

Secure Activation Function

We propose a new protocol that evaluates ρ from Figure 2(b) directly over additive shares and does not require full secure comparisons, which would have been more expensive. Instead of doing straightforward comparisons between z, 0.5 and -0.5, we derive the result through checking two things: (i) whether $z' = z + 1/2$ is positive or negative; (ii) whether $z' \geq 1$. Both checks can be performed without using a full comparison protocol.

When z' is bit decomposed, the most significant bit is 0 if z' is non-negative and 1 if z' is negative. In fact, if out of the λ bits, the a lowest bits are used to represent the fractional component and the b next bits are used to represent the integer component, then the remaining $\lambda - a - b$ bits all have the same value as the most significant bit. We will use this fact in order to optimize the protocol by only performing a partial bit-decomposition and deducting whether z' is positive or negative from the $(a + b + 1)$-th bit.

In the case that z' is negative, the output of ρ is 0. But, if z' is positive, we need to determine whether $z' \geq 1$ in order to know if the output of ρ should be fixed to 1 or to z'. A positive z' is such that $z' \geq 1$ if and only if at least one of the b bits corresponding to the integer component of z' representation is equal to 1, therefore we only need to analyze those b bits to determine if $z' \geq 1$.

Our secure protocol π_ρ is described in Protocol 5. The AND operation corresponds to multiplications in $\mathbb{Z}_2$. By the application of De Morgan's law, the OR operation is performed using the AND and negation operations. The successive multiplications can be optimized to only take a logarithmic number of rounds by using well-known techniques.

The activation function protocol π_ρ UC-realizes the activation function functionality $\mathcal{F}_\rho$. The correctness can be checked by inspecting the three possible cases: (i) if $z > 1/2$, then $\mathsf{pos} = 1$ and $\mathsf{geq1} = 1$ (since at least one of the bits representing the integer component of $z + 1/2$ will have a value 1). The output is thus $[\![2^a]\!]_{2^\lambda}$ (the fixed-point representation of 1); if $-1/2 \leq z < 1/2$, then $\mathsf{pos} = 1$ and $\mathsf{geq1} = 0$, and therefore the output will be $[\![z']\!]_{2^\lambda}$, which is the fixed-point representation of $z + 1/2$; if $z < -1/2$, then $\mathsf{pos} = 0$ and the output will be a secret sharing representing zero as expected. The security follows trivially from the UC-security of the building blocks used and the fact that no secret sharing is opened.

Functionality $\mathcal{F}_\rho$

Input: Upon receiving a message from Alice/Bob with her/his share of $[\![z]\!]_{2^\lambda}$, 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 z, compute the result of the activation function $\rho(z)$, and then create and distribute to Alice and Bob the secret sharing $[\![\rho(z)]\!]_{2^\lambda}$ (using the fixed-point representation). 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.

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 full gradient descent technique described in Algorithm 1, 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

Our secure LR training protocol $\pi_{\mathsf{LR-Training}}$ (described in Protocol 6) shows how the secure building blocks described before can be used 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

Protocol 5: Secure Protocol π_ρ for Computing the Activation Function ρ.

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 $\lambda > a + b$. Further, a negative value x is represented as $2^\lambda - |x|$.

```
Input  : $[\![z]\!]_{2^\lambda}$
Output: $[\![\rho(z)]\!]_{2^\lambda}$
$[\![z']\!]_{2^\lambda} \leftarrow [\![z]\!]_{2^\lambda} + 2^{a-1}$
$[\![z'_1]\!]_2, \dots [\![z'_{a+b+1}]\!]_2 \leftarrow \pi_{\mathsf{decomp}}([\![z']\!]_{2^\lambda} \wedge (2^{a+b+2} - 1))$
$[\![\mathsf{pos}]\!]_{2^\lambda} \leftarrow \pi_{\mathsf{2to2^\lambda}}(1 - [\![z'_{a+b+1}]\!]_2)$
$[\![\mathsf{geq1}]\!]_2 \leftarrow \bigvee_{i \in \{a+1,\dots,a+b\}} [\![z'_i]\!]_2$
$[\![\mathsf{geq1}]\!]_{2^\lambda} \leftarrow \pi_{\mathsf{2to2^\lambda}}([\![\mathsf{geq1}]\!]_2)$
$[\![r]\!]_{2^\lambda} \leftarrow 2^a [\![\mathsf{geq1}]\!]_{2^\lambda} + (1 - [\![\mathsf{geq1}]\!]_{2^\lambda})\, [\![z']\!]_{2^\lambda}$
$[\![\rho(z)]\!]_{2^\lambda} \leftarrow [\![\mathsf{pos}]\!]_{2^\lambda} [\![r]\!]_{2^\lambda}$
return $[\![\rho(z)]\!]_{2^\lambda}$
```

a fixed-point representation, we truncate the result using π_{trunc}. The activation function is securely computed using π_ρ on line 6. The multiplication of $t_d - o_d$ with $x_{d,i}$ is done using secure multiplication with batching on line 11. Since this also involves multiplication on numbers that are scaled, the result is truncated using π_{trunc} in line 14. A slight difference between the full gradient descent technique described in Algorithm 1 and our protocol $\pi_{\mathsf{LR-Training}}$, is that 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-$\pi_{\mathsf{decompOPT}}$ (see Appendix) within the activation function protocol π_ρ.

The LR training protocol $\pi_{\mathsf{LR-Training}}$ UC-realizes the logistic regression training functionality $\mathcal{F}_{\mathsf{LR-Training}}$. The correctness is trivial and the security follows straightforwardly from the UC-security of the building blocks used in $\pi_{\mathsf{LR-Training}}$.

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

Input: Upon receiving a message from Alice/Bob with her/his shares of $([\![\boldsymbol{x}_d]\!], [\![t_d]\!])$ for the set of training examples D, 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, locally perform the same computational steps as π_ρ using the secret sharings. Let $[\![\boldsymbol{w}]\!]$ be the resulting vector. Before the deliver of the output shares, a corrupt party can fix the shares that it will get, in which case the other shares are adjusted accordingly to still sum to $\boldsymbol{w}$. The output shares are delivered to the parties.

Protocol 6: Secure Logistic Regression Training Protocol $\pi_{\mathsf{LR-Training}}$.

Input : $([\![\boldsymbol{x}_d]\!], [\![t_d]\!])$ for a set of training examples D; learning rate η; number of iterations n_{iter}. All secret sharings in the description of this protocol are in $\mathbb{Z}_{2^\lambda}$ and thus we simplify the notation to $[\![\cdot]\!]$.

Output: $[\![\boldsymbol{w}]\!]$ for a vector of weights w_i that minimize the sum of squared errors over the training data

```
1  for $i \leftarrow 0$ to $m$ do
2      $[\![w_i]\!] \leftarrow 0$
3  for 0 to $n_{iter}$ do
4      for each $(\boldsymbol{x}_d, t_d)$ in $D$ do
5          $[\![z_d]\!] \leftarrow \pi_{\mathsf{trunc}}(\pi_{\mathsf{IP}}([\![\langle \boldsymbol{w} \rangle]\!], [\![\langle \boldsymbol{x}_d \rangle]\!]))$
6          $o_d \leftarrow \pi_\rho([\![z_d]\!])$
7      for $i \leftarrow 0$ to $m$ do
8          $[\![\Delta w_i]\!] \leftarrow 0$
9      for each $(\boldsymbol{x}_d, t_d)$ in $D$ do
10         $[\![\boldsymbol{v}_{\mathsf{diff}}]\!] \leftarrow \langle ([\![t_d]\!] - [\![o_d]\!]), \dots, ([\![t_d]\!] - [\![o_d]\!]) \rangle$ // vector of length $m$
11         $[\![\boldsymbol{v}_{\mathsf{gradient}}]\!] \leftarrow$batch-$\pi_{\mathsf{DM}}([\![\boldsymbol{v}_{\mathsf{diff}}]\!], [\![\boldsymbol{x}_d]\!])$
12         $[\![\Delta \boldsymbol{w}]\!] \leftarrow [\![\Delta \boldsymbol{w}]\!] + [\![\boldsymbol{v}_{\mathsf{gradient}}]\!]$
13     for $i \leftarrow 0$ to $m$ do
14         $[\![\Delta w_i]\!] \leftarrow \pi_{\mathsf{trunc}}([\![\Delta w_i]\!])$
15         $[\![w_i]\!] \leftarrow [\![w_i]\!] + \eta [\![\Delta w_i]\!]$
16 return $[\![\boldsymbol{w}]\!]$
```

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

1 The TI sends 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.

2 Each data owner converts the values in the set of training examples D that it holds 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 respectively.

3 Each data processor receives the shares of data from the data owners. They now have secret sharings $(\llbracket \boldsymbol{x}_d \rrbracket, \llbracket t_d \rrbracket)$ of the set of training examples D. The learning rate η and number of iterations n_{iter} are predetermined and public to both data processors.

4 The data processors collaborate to train the LR model. They both follow the secure LR training protocol $\pi_{\mathsf{LR-Training}}$.

5 At the end of the protocol, each data processor will hold shares of the model's weights $\llbracket w_i \rrbracket$. Each data processor sends their shares to all of the data owners, who can then combine the shares to learn the weights of the LR model.

Cryptographic Engineering Optimizations

Sockets and Threading

A single iteration of the LR protocol is highly parallelizable 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. We implemented the proposed protocols in Rust; 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.

Results

We implemented the protocols from the **Methods** section in Rust[4] and experimentally evaluated them on the BC-TCGA and GSE2034 data sets of the iDASH 2019 competition. Both data sets contain gene expression data from breast cancer patients which are normal tissue/non-recurrence samples (negative) or breast cancer tissue/recurrence tumor samples (positive) [36]. We trained LR models on both data sets with a learning rate $\eta = 0.001$. We use a fixed number of iterations for each data set: 10 iterations for the BC-TCGA data set and 223 iterations for the GSE2034 data set. The accuracy of the resulting models, evaluated with 5-fold cross-validation, is presented in Table 1, along with the average runtime 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 previous section). 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 [36] on the BC-TCGA and GSE2034 data sets within about 2.52 seconds and 26.90 seconds respectively.

A previous version of this implementation was submitted to the iDASH 2019 Track 4 competition. 9 of the 67 teams who entered Track 4 completed the challenge. Our solution was one of the 3 solutions who tied for the first place. Our implementation trained on all of the features for both data sets (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. The implementation presented in the current work is further optimized in relation to the iDASH version and achieves far better runtimes.

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. While SecureML does not originally use a TI to predistribute the multiplication triples, it would be easy to adapt their result to use a TI for that purpose. Therefore, in order to have a fair comparison, we compare our protocol runtime against only their

[4] https://bitbucket.org/uwtppml/idash2019

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%	2.52 sec
GSE2034	12,634	142	83	223	64.82%	26.90 sec

Table 2 Runtime comparisons between SecureML and our work

	BC-TCGA training (online)	GSE2034 training (online)	activation function (one evaluation)
Our work	2.52 sec	26.90 sec	0.030 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	9 ms	0.035 ms
512	16 ms	0.031 ms
1024	30 ms	0.029 ms
2048	59 ms	0.028 ms

online runtime (thus excluding their offline runtime). We evaluated our implementation's runtime against SecureML's implementation by running their implementation on the same AWS machines using the same data sets (see Table 2 for runtime comparisons). For both data sets, 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 30 ms (0.029 ms per activation function). This makes our secure activation function implementation nearly twice as fast as SecureML's. 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 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 data set that were part of the 76-gene signature described in [37]. 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 0.51 seconds, which is about a 2 second decrease from training on all 17,814 features. The genes in the GSE2034 data set 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 6.71 seconds, which is about a 20 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 LR 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 avoids the use of full-fledged secure comparison protocols; (ii) a series of cryptographic engineering optimizations to improve the performance.

With our implementation, we can train on the BC-TCGA data set with 17,814 features and 375 samples with 10 iterations in 2.52 seconds, and we can train on the GSE2034 data set with 12,634 features and 179 samples with 223 iterations in 26.90 seconds. A less optimized version of this implementation won first place at the iDASH 2019 Track 4 competition when considering accuracy and efficiency. Our solution is particularly efficient for LANs where we can perform 1024 secure computations of the activation function in about 30 ms. To the best of our knowledge, ours is the fastest protocol for privately training logistic regression models over local area networks.

List of abbreviations

ML: Machine Learning; MPC: Multi-Party computation; LR: Logistic regression; UC: Universal composability; MSB: Most significant bit; TI: Trusted initializer; LAN: Local area network

Appendix

Optimization of π_{decomp}

Overview and Previous Work

The functionality $\mathcal{F}_{\mathsf{decomp}}$ (described in Section **Methods**) is easily realized as an adder circuit that takes as inputs each bit of the additive shares of a secret sharing $[\![x]\!]_{2^\lambda}$ in a large ring $\mathbb{Z}_{2^\lambda}$ and outputs an "XOR-sharing" of the secret $[\![x_1]\!]_2, \ldots, [\![x_\lambda]\!]_2$. First, each party regards its share of $[\![x]\!]_{2^\lambda}$, denoted x_i, as an XOR-shared secret $[\![x_{i,1}]\!]_2, \ldots, [\![x_{i,\lambda}]\!]_2$ and passes it to the adder circuit. The adder circuit then computes the *carry vector* which accounts for the rollover of binary addition. Adding this vector to all bitwise shares $[\![x_{1,j}]\!]_2, [\![x_{2,j}]\!]_2$ resolves the difference between $[\![x_{1,1}]\!]...[\![x_{1,\lambda}]\!] \oplus [\![x_{2,1}]\!]...[\![x_{2,\lambda}]\!]$ and the bit-decomposed secret x.

Naively, this carry vector can be obtained with linear communication complexity by means of ripple carry addition, as is described in Protocol 3. But, it is possible to achieve logarithmic communication complexity and even constant complexity [38] (though with worse performance than the logarithmic version for all reasonable bit lengths).

The highest performing realization of $\mathcal{F}_{\mathsf{decomp}}$ for realistic bit lengths is based on a speculative adder circuit [26] in which at each layer the next set of carry bits are computed twice; once for each case that the previous carry bit had been 0 and 1. This protocol has $\lceil log(\lambda)\rceil + 2$ rounds of communication and requires a total data transfer of $4\lambda\lceil log(\lambda)\rceil + 6\lambda$ bits.

We propose a new, highly optimised protocol based on a matrix composition network that reduces the number of communication rounds by 1 (or 2, in special cases) and requires a small fraction of the aforementioned data transfer cost.

Matrix composition network

To sum the binary numbers a and b, the i-th bit is given by $s_i = a_i \oplus b_i \oplus c_{i-1}$, where $c_i = a_ib_i \oplus a_ic_{i-1} \oplus b_ic_{i-1}$. In an alternate view, the carry can be seen to depend on two signals which in turn depend on a and b. Generate ($g_i = a_ib_i$) creates a new carry bit at the i-th position, and Propogate ($p_i = a_i \oplus b_i$) perpetuates the previous carry bit, if it exists. In this representation, $s_i = p_i \oplus c_{i-1}$ and $c_i = g_i + p_ic_{i-1}$. This sum-of-products form of the expression for c_i lends itself to a matrix representation

$$\begin{bmatrix} c_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_i & g_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix} = M_i \begin{bmatrix} c_{i-1} \\ 1 \end{bmatrix}.$$

When matrices in the form of M_i are composed, the lower entries remain unchanged. This implies that

$$\begin{bmatrix} c_i \\ 1 \end{bmatrix} = \begin{bmatrix} p_i & g_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_{i-1} & g_{i-1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} c_{i-2} \\ 1 \end{bmatrix} = M_iM_{i-1} \begin{bmatrix} c_{i-2} \\ 1 \end{bmatrix}.$$

Therefore, to compute all c_i, it is sufficient to compute the set of all matrix compositions

$$\left\{ \prod_{j=1}^{i} M_j \,\middle|\, 1 \leq i < \lambda \right\}.$$

Note that it is not necessary to compute the λ-th carry bit because s_λ depends on $c_{\lambda-1}$. Treating the carry-in to the 1st bit as the vector $(0, 1)$, all c_i can be derived implicitly from the upper right-hand entry of $M_{1.i}$ (here, $M_{1,i}$ denotes the matrix composed of all matrices M_1 through M_i, consecutively).

From the MPC perspective, this matrix composition requires two $\mathbb{Z}_2$ multiplications: $p_{i+1}p_i$ and $p_{i+1}g_i$ as seen in the equation below. The OR operation (+), which usually requires multiplication in MPC, is reduced to XOR based on the observation that p_{i+1} and g_{i+1} cannot both be true for a given i.

$$\begin{bmatrix} p_{i+1} & g_{i+1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_i & g_i \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} p_{i+1}p_i & p_{i+1}g_i + g_{i+1} \\ 0 & 1 \end{bmatrix}$$

The entire set of matrix compositions can be realized in a logarithmic depth network by, at the i-th layer, computing all compositions $M_{1.j}$ that require fewer than 2^{i-1} compositions. To set up conditions to allow us to minimize the total data transfer, the constraint is added that each $M_{1.j}$ should be the composition of the "largest" matrix from the previous layer, $M_{1.2^{i-2}}$, with the remainder $M_{2^{i-2}+1.j}$. If $M_{2^{i-2}+1.j}$ doesn't exist in the network, it is added recursively following the same set of constraints.

Figure 4 shows an example with $\lambda = 17$. This network is hereafter referred to as $\mathsf{ComposeNet}_p$ where p is the highest order bit to decompose. The protocol description that follows considers only the case where $p = \lambda$, though the protocol functions the same for any $p \leq \lambda$. For instance, in Protocol 3, when using π_{decomp} to find the MSB of a secret, it is sufficient to set $p = a + b + 1$.

Protocol 7: Secure Protocol $\pi_{\mathsf{decompOPT}}$ for computing $\mathcal{F}_{\mathsf{decomp}}$ more efficiently.

```
Input  : $[\![x]\!]_{2^\lambda}$
Output: $[\![x_1]\!]_2 ... [\![x_\lambda]\!]_2$
Party $i$ regards its share $x_i$ as $p_{i,1}, \dots, p_{i,\lambda}$ s.t.
$[\![p_j]\!]_2, = p_{1,j} \oplus p_{2,j}$ for $j = 1, \dots, \lambda$
Party 1 creates the sharing $[\![g_{1,j}]\!]_2 = (p_{1,j}, 0)$.
Party 2 creates the sharing $[\![g_{2,j}]\!]_2 = (0, p_{2,j})$.
$[\![g_j]\!]_2 \leftarrow [\![g_{1,j}]\!]_2 [\![g_{2,j}]\!]_2$
$[\![M_j]\!]_2 \leftarrow \begin{bmatrix} [\![p_j]\!]_2 & [\![g_j]\!]_2 \\ 0 & 1 \end{bmatrix}$ for all $j$
$\{[\![M_{1.j}]\!]_2 | 1 \leq j < \lambda\} \leftarrow \mathsf{ComposeNet}_\lambda([\![M]\!]_2)$
$[\![c_j]\!]_2 \leftarrow$ the upper right entry of $[\![M_{1.j}]\!]_2$
$[\![s_1]\!]_2 \leftarrow [\![p_1]\!]_2$
$[\![s_j]\!]_2 \leftarrow [\![p_j]\!]_2 \oplus [\![c_{j-1}]\!]_2$ for all $j > 1$
return $[\![s_1]\!]_2 ... [\![s_\lambda]\!]_2$
```

Efficiency discussion

The setup phase prior to the call to $\mathsf{ComposeNet}_\lambda$ requires λ multiplications over $\mathbb{Z}_2$ to compute all $[\![g_j]\!]$. This corresponds to one communication round and 2λ bits of data transfer.

A call to $\mathsf{ComposeNet}_\lambda$ has communication complexity corresponding to the depth of the network, $\lceil \log(\lambda - 1) \rceil$, and $\frac{\lambda}{2}$ multiplications over $\mathbb{Z}_2$ per layer, with fewer on the final layer when $\lambda - 1$ is not a power of 2. However, due to the fact that the matrices at each node of $\mathsf{ComposeNet}_\lambda$ are reused extensively and known to not change value, the Beaver Triples used to mask the matrices can be desgined to contain redundancies to minimise the data transfer at each layer [3]. By re-using correlated randomness where information leakage is not possible, only $\frac{\lambda}{2} - (2^{i-i} - 1)$ masks need to be transferred at depth i, for $i > 0$. At depth 0, there are λ masks; one for each matrix. Each matrix mask is 2 bits (one for each of the Propogate and Generate bits), so the total data transfer is $2\lambda + 2\sum_{i=1}^{\lceil log(\lambda-1) \rceil - 1} (\frac{\lambda}{2} + 1 - 2^{i-1})$.

The recombination phase after $\mathsf{ComposeNet}_\lambda$ is computed has only local computations and thus contributes nothing to the complexity.

Combining all phases, we see that $\pi_{\mathsf{decompOPT}}$ has a communication cost of $\lceil \log(\lambda - 1) \rceil + 1$ and a total data transfer cost of $4\lambda + 2\sum_{i=1}^{\lceil \log(\lambda-1) \rceil - 1} (\frac{\lambda}{2} + 1 - 2^{i-1})$ bits. Comparing with the speculative adder's performance, the number of communication rounds is decreased by 1 in all cases and 2 in the case that $\lambda - 1$ is a power of 2. The total data transfer cost has roughly $\frac{1}{3}$ the data transfer rate of the previous work at $\lambda = 8, 16$. For higher all bit lengths, the ratio quickly converges near $\frac{1}{4}$.

Implementation and Batching

$\mathsf{ComposeNet}_\lambda$ can be implemented efficiently as a set of index pairs that correspond to the positions of the Propogate and Generate bits that need to be combined at each layer. Once per layer, all products $p_{i+1}p_i$, $p_{i+1}g_i$ can be computed in a single call to π_{DM} by taking the bitwise product between the concatenations $p_{i+1}||p_{i+1}$, $p_i||g_i$ and splitting the result.

Extending to the case that many values need to be bit decomposed at the same time (as in Protocol 6), a vector of inputs can be decomposed "in parallel" by taking vertical slices over the Generate and Propogate bits of each element and re-packing them into a transposed form. In this way, each layer of $\mathsf{ComposeNet}_\lambda$ can operate on a vector of matrices (represented as two lists of bit slices) to produce a vector of matrix compositions. This method has no effect on the number of rounds of communication and the total data transfer scales linearly with the length of the input vector.

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Availability of data and materials

The genomic data set was available upon request during the iDASH 2019 competition.
https://iu.app.box.com/s/6pbynxgscyxl7facstigb8w6jc17o99z

Competing interests

The authors declare that they have no competing interests.

Funding

Rafael Dowsley is supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

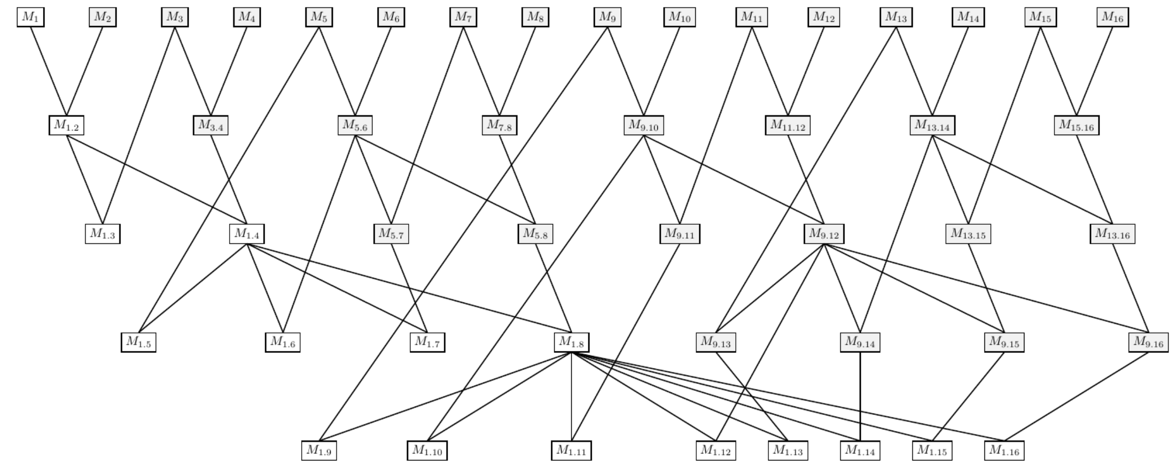


Figure 4 ComposeNet$_\lambda$ for $\lambda = 17$. Computes the set of all matrix compositions $M_1, M_{1.2}, M_{1.3}, \ldots, M_{1.(\lambda-1)}$. The notation $M_{i.j}$ means "the composition of all matrices i through j." The greyed nodes are only used for intermediate computations and the white nodes are part of the solution set.

Authors' contributions

All authors worked together on the overall design of the solution in the clear and in private. MDC proposed the use of LR and derived the gradient descent algorithm for minimizing the sum of squared errors with a neuron with a clipped ReLu activation function. 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

The authors want to thank P. Mohassel for making the SecureML code available that was used for the experimental comparison in the Section **Results**.

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. De Cock M, Dowsley R, Nascimento A, Railsback D, Shen J, Todoki A. Fast Secure Logistic Regression for High Dimensional Gene Data. In: Privacy in Machine Learning (PriML2019). Workshop at NeurIPS; 2019. p. 1–7.
3. 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.
4. Bonte C, Vercauteren F. Privacy-preserving logistic regression training. BMC Medical Genomics. 2018;11(4):86.
5. 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.
6. 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.
7. 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.
8. 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).
9. Xie W, Wang Y, Boker SM, Brown DE. Privlogit: Efficient privacy-preserving logistic regression by tailoring numerical optimizers. arXiv preprint arXiv:161101170. 2016;p. 1–25.
10. Wagh S, Gupta D, Chandran N. SecureNN: 3-Party Secure Computation for Neural Network Training. Proceedings on Privacy Enhancing Technologies. 2019;1:24.
11. 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. IEEE Computer Society; 2001. p. 136–145.
12. Cramer R, Damgård I, Nielsen JB. Secure Multiparty Computation and Secret Sharing. Cambridge University Press; 2015.
13. Canetti R, Fischlin M. Universally Composable Commitments. In: Kilian J, editor. Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings. vol. 2139 of Lecture Notes in Computer Science. Springer; 2001. p. 19–40.
14. Canetti R, Lindell Y, Ostrovsky R, Sahai A. Universally composable two-party and multi-party secure computation. In: Reif JH, editor. Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada. ACM; 2002. p. 494–503.
15. Peikert C, Vaikuntanathan V, Waters B. A Framework for Efficient and Composable Oblivious Transfer. In: Wagner DA, editor. Advances in Cryptology - CRYPTO 2008, 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings. vol. 5157 of Lecture Notes in Computer Science. Springer; 2008. p. 554–571.
16. Barak B, Canetti R, Nielsen JB, Pass R. Universally Composable Protocols with Relaxed Set-Up Assumptions. In: 45th Symposium on Foundations of Computer Science (FOCS 2004), 17-19 October 2004, Rome, Italy, Proceedings. IEEE Computer Society; 2004. p. 186–195.

17. Hofheinz D, Müller-Quade J. Universally Composable Commitments Using Random Oracles. In: Naor M, editor. Theory of Cryptography, First Theory of Cryptography Conference, TCC 2004, Cambridge, MA, USA, February 19-21, 2004, Proceedings. vol. 2951 of Lecture Notes in Computer Science. Springer; 2004. p. 58–76.
18. Barreto PSLM, David B, Dowsley R, Morozov K, Nascimento ACA. A Framework for Efficient Adaptively Secure Composable Oblivious Transfer in the ROM. IACR Cryptology ePrint Archive. 2017;2017:993.
19. Dowsley R, Müller-Quade J, Nascimento ACA. On the Possibility of Universally Composable Commitments Based on Noisy Channels. In: SBSEG 2008. Gramado, Brazil; 2008. p. 103–114.
20. Dowsley R, van de Graaf J, Müller-Quade J, Nascimento ACA. On the Composability of Statistically Secure Bit Commitments. Journal of Internet Technology. 2013;14(3):509–516.
21. Katz J. Universally Composable Multi-party Computation Using Tamper-Proof Hardware. In: Naor M, editor. Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings. vol. 4515 of Lecture Notes in Computer Science. Springer; 2007. p. 115–128.
22. Dowsley R, Müller-Quade J, Nilges T. Weakening the Isolation Assumption of Tamper-Proof Hardware Tokens. In: Lehmann A, Wolf S, editors. ICITS 15: 8th International Conference on Information Theoretic Security. vol. 9063 of Lecture Notes in Computer Science. Lugano, Switzerland: Springer, Heidelberg, Germany; 2015. p. 197–213.
23. 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.
24. 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.
25. 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.
26. 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.
27. Reich D, Todoki A, Dowsley R, De Cock M, Nascimento ACA. Privacy-Preserving Classification of Personal Text Messages with Secure Multi-Party Computation. In: Wallach HM, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox EA, Garnett R, editors. Advances in Neural Information Processing Systems 32 (NeurIPS); 2019. p. 3752–3764.
28. 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.
29. 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.
30. 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.
31. 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.
32. 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.
33. 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.
34. Beaver D. Commodity-based cryptography. In: STOC. vol. 97; 1997. p. 446–455.
35. Dowsley R. Cryptography Based on Correlated Data: Foundations and Practice. Karlsruhe Institute of Technology, Germany; 2016.
36. 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.
37. 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.
38. Toft T. Constant-Rounds, Almost-Linear Bit-Decomposition of Secret Shared Values. In: Topics in Cryptology - CT-RSA 2009, The Cryptographers' Track at the RSA Conference 2009, San Francisco, CA, USA, April 20-24, 2009. Proceedings; 2009. p. 357–371.