Privacy-Preserving Text Classification with Secure Multiparty Computation

by

Devin Reich

Supervised by Martine De Cock

A senior thesis submitted in partial fulfillment of the departmental honors requirements for the degree of

Bachelor of Science
Computer Science & Systems
University of Washington Tacoma

March 2020

Presentation of work given at NeurIPS2019 on December 10, 2019

The student has satisfactorily completed the Senior Thesis, presentation and senior elective course requirements for CSS Departmental Honors.

Faculty advisor: [illegible] Date Apr 3, 2020

CSS Program Chair: Donald Chinn Date April 3, 2020

©Copyright 2020
Devin Reich

Privacy-Preserving Text Classification with Secure Multiparty Computation

Devin Reich

An honors thesis
submitted in partial fulfillment of the
requirements for

Computer Science and Systems Honors

University of Washington Tacoma

2020

Reading Committee:

Dr. Martine De Cock, Chair

Dr. Anderson Nascimento

Program Authorized to Offer Degree:
B.S. in Computer Science and Systems

University of Washington Tacoma

Abstract

Privacy-Preserving Text Classification with
Secure Multiparty Computation

Devin Reich

Chair of the Supervisory Committee:
Dr. Martine De Cock
School of Engineering and Technology

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 implemented our method as part of an open source framework for privacy-preserving machine learning, in the programming languages Java and Rust. We present end-to-end experiments and a comparative evaluation of the implementations in Java and Rust, with an application for detecting hate speech against women and immigrants, demonstrating excellent runtime results without loss of accuracy. As such, our work shows that securely classifying unstructured text is feasible in real world applications.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Dr. Anderson C.A. Nascimento and Dr. Martine De Cock, to whom I credit with my success. I also wish to thank Ariel Todoki for her help implementing secure logistic regression and AdaBoost models as well as Rafael Dowsley for writing security proofs to show that our implementation is provably secure and correct. Additionally, I extend my thanks to Davis Railsback for his help with the Rust Lynx repository as he wrote the basic architecture of the framework and helped describe the language and the framework usage, jump starting this thesis' implementation in Rust and reducing the learning curve for the language of which I had no prior experience.

Chapter 1
INTRODUCTION

With the drastic increase in the availability of data and the increase in computational power that enables the processing of this newly available data, the utility of machine learning (ML) is unrestricted. For example, healthcare providers use ML to help doctors identify whether a tumor is benign or malignant. ML can also identify whether or not a tweet contains hate speech or not; which in turn can help filter out what a user sees. These are prime examples of how ML has the power to save lives and improve user experience along with countless other uses.

While ML is extremely powerful, it comes with an Achilles heel: data. In supervised ML, there are two phases: training and scoring. Both the training of the ML model and using the trained model to score – or classify – new data points requires access to data. In this thesis we focus specifically on *text classification*, namely assigning an unstructured text to a category with a pre-trained ML classifier. This unstructured text may be personal in nature, such as medical records, text messages, or personal e-mails, that need to be classified according to respectively ICD10 codes (for health insurance purposes) [39], or to infer whether the author is depressed [44], suicidal [40], a terrorist threat [2], or whether the e-mail is a spam message [3, 47]. Other valuable applications of text message classification include user profiling for tailored advertising [29], detection of hate speech [6], and detection of cyberbullying [49]. Some of the above are integrated in parental control applications[1] 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

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

this thesis, we propose the first privacy-preserving solution for *unstructured* text classification that is provably secure in the honest-but-curious setting.

We consider the scenario where there are two parties, *Alice* and *Bob* (see Figure 1.1). Alice has raw text that needs to be classified while Bob has the ML model to score the text. If Alice were to give her text directly to Bob to be scored, there exists a breach of privacy. Instead, Bob could give his ML model to Alice and let her score her text. This poses another privacy concern: what if the model being used in the computations is also private? The latter is particularly relevant when Bob's classifier constitutes a competitive advantage, or in security applications such as spam or hate speech detection, where knowledge of the model would help adversaries to develop strategies for evading detection.

In this thesis, we propose a secure text classification protocol that 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. To this end, we use Secure Multiparty Computation (SMC) [15], a technique in cryptography that has successfully been applied to various ML tasks with *structured* data (see e.g. [13, 18, 20, 38] and references therein). To the best of our knowledge there are no existing differential privacy (DP) or SMC based solutions for privacy-preserving 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 [14] while leaking information about the text being classified.

To demonstrate the feasibility of our proposed protocol, we perform end-to-end experiments with an application for privacy-preserving 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 deep learning architectures for hate speech detection [33]. 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

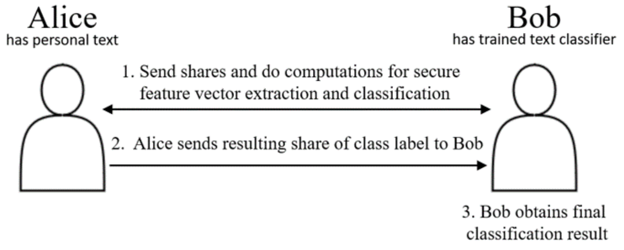
Alice
has personal text
Bob
has trained text classifier
1. Send shares and do computations for secure feature vector extraction and classification
2. Alice sends resulting share of class label to Bob
3. Bob obtains final classification result

Figure 1.1: Roles of Alice and Bob in SMC based text classification

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 [14].

We build our protocols using a privacy-preserving machine learning (PPML) framework based on SMC developed at UW Tacoma[2]. 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 privacy-preserving text classification.

This thesis is structured as follows. In Chapter 2 we recall the necessary preliminaries regarding text classification in the clear, i.e. without encryption. We present the dataset and the trained text classifiers that we use later in the thesis to demonstrate the feasibility of our proposed protocol for secure text classification. In Chapter 3, we introduce the preliminary background on SMC, including the fundamentals of additive secret sharing, and descriptions of the cryptographic primitives that we use as building blocks in our secure text classification protocol. In Chapter 4, we discuss privacy-preserving text classification and we present the

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

novel protocols that we designed to accomplish this feat. In Chapter 5, we provide details about two ways in which we implemented the protocols proposed in Chapter 4, namely in the programming language Java, and in the programming language Rust. We find that, with our implementations, we can securely classify tweets within seconds, and that the fastest implementation in the Rust-Lynx framework is 5 times faster than that in Java-Lynx. In Chapter 6, we summarize our conclusions and lay out interesting directions for future work. A correctness and security analysis of the protocols presented in this thesis is provided as an appendix.

Part of this thesis has been published as [45]:

> **D. Reich**, A. Todoki, R. Dowsley, M. De Cock, A. Nascimento. Privacy-Preserving Classification of Personal Text Messages with Secure Multi-Party Computation. Advances in Neural Information Processing Systems 32 (NeurIPS), p. 3752-3764, 2019

Chapter 2
MACHINE LEARNING FOR TEXT CLASSIFICATION IN THE CLEAR

2.1 Data Set

In Chapter 5, we evaluate the protocols designed for this thesis in a privacy-preserving application for detecting whether or not a tweet contains hate speech. The first portion of the work was to train models on the 2019 SemEval hate speech data set [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.

2.2 Models

We use logistic regression (LR) and AdaBoost (AB) ML model architectures with feature selection. We evaluated the two model architectures chosen, LR and AB, on various feature spaces of k = 50, 200, and 500 features. The features were chosen by taking the top k features of a random forest trained on the entire data set, one for unigrams only and one for unigrams and bigrams combined.

2.2.1 Logistic Regression

A logistic regression (LR) model is a binary classifier that models the posterior probability $P(Y|X)$ of the class Y given the input feature vector X by fitting a logistic curve (sigmoid function) to the relationship between X and Y. The probability of class 1 and 0 are respectively computed as:

$$P(Y = 1|X) = \frac{1}{1 + \exp\left(w_0 + \sum_{i=1}^{n} w_i \cdot X_i\right)} \tag{2.1}$$

$$P(Y = 0|X) = \frac{\exp\left(w_0 + \sum_{i=1}^{n} w_i \cdot X_i\right)}{1 + \exp\left(w_0 + \sum_{i=1}^{n} w_i \cdot X_i\right)} \tag{2.2}$$

where $w_0, w_1, \ldots, w_n$ are model parameters (weights) that are learned from training data. To obtain the label $Y = 0$, we need to satisfy:

$$1 < \frac{P(Y = 0|X)}{P(Y = 1|X)} \tag{2.3}$$

This simplifies to:

$$1 < \exp\left(w_0 + \sum_{i=1}^{n} w_i \cdot X_i\right) \tag{2.4}$$

Taking the log of both sides, we get:

$$0 < w_0 + \sum_{i=1}^{n} w_i \cdot X_i \tag{2.5}$$

Expression (2.5) can be used directly to perform inference (classification) with a trained logistic regression model: if, for a given feature vector X and learned weights $w_0, w_1, \ldots, w_n$, the expression in the right hand side of (2.5) evaluates to a strictly positive number, then the model assigns class label 0 to the instance, and class label 1 otherwise. It is this principle that we leverage later in this thesis for secure classification with a trained logistic regression model, replacing the computation on the right hand side of (2.5) by doing a secure dot product (see Chapter 4).

2.2.2 AdaBoost

An AdaBoost model is an ensemble or set of decision trees that are trained in sequence [30]. As with logistic regression, an Adaboost model can be used for classification, i.e. to map an input vector $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ consisting of n features (in our case: word unigrams and bigrams) to a class label $c_1, \ldots, c_t$. To this end, first the input vector $\mathbf{x}$ is classified with each individual decision tree in the ensemble. Each internal node of the tree structure tests the value of a particular feature against a corresponding threshold and branches according to the results. Each leaf node specifies one of the t classes. The result of the classification based on an individual decision tree is the class associated with the leaf reached from traversing the tree. In our case, each internal node checks whether a particular unigram or bigram is present in the tweet or not, and there are $t = 2$ class labels, namely whether the tweet contains hate speech or not.

The decision trees in an Adaboost model are often shallow. In this thesis, we follow the common practice of using decision tree stumps, i.e. decision trees of depth 1 (a root and a leaf level). When decision trees are not grown to full depth during training, it is natural to associate each leaf with a probability of the t classes. In our case, this means that each leaf contains a probability that the tweet contains hate speech, and a probability that the tweet does not contain hate speech, with both probabilities in a leaf summing up to 1. Using $k = 50$, 200, and 500 features, we create AB models of k binary stumps where each leaf has a probability for the positive and negative class. In addition, during the training process, a confidence factor or weight is learned for each decision stump.

During classification, the input vector $\mathbf{x}$ is classified with each of the k decision stumps, resulting in k inferred probability distributions. These k probability distributions are aggregated by taking a weighted sum, where each probability distribution is weighted with the confidence factor of the tree that produced it. This means that certain decision stumps (features) have more impact on the decision of the classification than others. The final predicted class label is the class label with the highest weighted sum of probabilities.

2.3 Results

Accuracy results for a variety of models trained to classify a tweet as hate speech vs. non-hate speech are presented in Table 2.1. The models are evaluated using 5-fold cross-validation over the entire corpus of 10,000 tweets. The top rows in Table 2.1 correspond to tree ensemble models consisting of 50, 200, and 500 decision stumps respectively; the root of each stump

Table 2.1: Accuracy (Acc) results using 5-fold cross-validation over the corpus of 10,000 tweets.

	Unigrams	**Unigrams+Bigrams**
	Acc	**Acc**
Ada; 50 trees; depth 1	71.6%	73.3%
Ada; 200 trees; depth 1	73.0%	74.2%
Ada; 500 trees; depth 1	73.9%	74.4%
Logistic regression (50 feat.)	72.4%	73.8%
Logistic regression (200 feat.)	73.3%	73.7%
Logistic regression (500 feat.)	73.4%	74.2%

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). 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 thesis. Instead, our goal is to demonstrate that privacy-preserving text classification based on SMC is feasible in practice.

Chapter 3

SECURE MULTIPARTY COMPUTATION AND CRYPTOGRAPHIC PRIMITIVES

This chapter will cover the basics of SMC, preliminary abstract algebra knowledge, and the encryption method we use. The description of the encryption will cover the basic cryptographic building blocks of addition and multiplication and the basic protocols we have built using circuits of additions and multiplications.

3.1 Secure Multiparty Computation

3.1.1 Informal Definition

We consider *honest-but-curious adversaries*, as is common in SMC based PPML (see e.g. [18, 20]). An honest-but-curious adversary follows the instructions of the protocol, but tries to gather additional information. Secure protocols prevent the latter.

In SMC, parties work together to jointly compute a function that takes in n private inputs defined by $f(x_1, x_2, \ldots, x_n)$. Each input $x_1 \ldots x_n$ is encrypted, using additive secret shares in this thesis, which prevents the parties from learning information about the inputs. Similarly, the intermediate calculations and the output is also secret shared which means the data is encrypted from end to end. Note that each function described uses an arbitrary n inputs to the function. We reuse the variable n as to denote the length of the input vector but each function's input vector length is arbitrary and independent of the other functions described in the thesis.

3.1.2 Shares

All computations used in this thesis are done over additive secret shares, which is our choice of encryption for SMC. All shares are integers belonging to the general set, $\{0, 1, \ldots, m-1\}$, defined as $\mathbb{Z}_m$, where m is the modulus that defines the set. When the integer is a binary value, we specifically denote that it belongs to the set $\{0, 1\}$, defined as $\mathbb{Z}_2$. Given an integer X, where $X \in \mathbb{Z}_m$, we split X into shares such that the shares X_i satisfy the equation $X = \sum_{i=1}^{k} X_i \mod m$. Each share X_i is generated at random to ensure that any particular share reveals no information about X. This scheme works for k parties and the parties can reveal X by summing each $X_i \mod m$.

We break down all protocols into circuits of secure multiplications and additions. If we can successfully break down a function into secure multiplications and additions, then the whole function will be secure. Note from this point on, we will consider the case where $k = 2$ as our framework was tested with 2 parties and the examples are easier to write with only two parties; however, our protocols extend to an arbitrary k parties.

3.1.3 Asymmetric Bit

Out of the k parties, one party will be designated to have an asymmetric bit, defined 1 if the party has it; otherwise the value is a 0. We will denote this bit as $\hat{b}$. This bit allows us to add constants and perform protocols that only one party needs to do a computation for. In our two party computations, we will let Alice be the party with the asymmetric bit.

3.1.4 Secure Addition

Alice and Bob start with random shares X_A, Y_A, X_B and Y_B that are respectively Alice and Bob's shares of X and Y. Alice and Bob can locally add up their shares of X and Y. When recombined, the actual sum of X and Y is revealed.

Example 1. Let $X = 2$, $Y = 3$ and $m = 7$. Let $X_A = 4$, $X_B = 5$ which satisfies $X = X_A + X_B \mod m$ and let $Y_A = 1$, $Y_B = 2$ which satisfies $Y = Y_A + Y_B \mod m$. Then Alice and Bob can add up

$$Z_A = (X_A + Y_A \mod m) = (4 + 1 \mod m) = 5$$

and

$$Z_B = (X_B + Y_B \mod m) = (5 + 2 \mod m) = 0$$

To reveal Z, they can compute $5 + 0 \mod m = 5$. Notice in the clear $Z = X + Y = 2 + 3 = 5$. Therefore, Alice and Bob securely computed Z without revealing the actual values of X and Y.

3.1.5 Secure Multiplication and Beaver Triples

The second secure building block is secure multiplication. Consider Alice and Bob have the same X and Y and the same shares of X and Y as in the previous example. They want to compute

$$\begin{aligned} Z &= X \cdot Y \\ &= (X_A + X_B) \cdot (Y_A + Y_B) \\ &= X_A \cdot Y_A + X_A \cdot Y_B + X_B \cdot Y_A + X_B \cdot Y_B \end{aligned}$$

Notice the middle terms $X_A \cdot Y_B$ and $X_B \cdot Y_A$ require that either Alice or Bob gives up one their shares in order to make this computation. This means the protocol as is defined above is insecure because whoever receives the share could recover the actual value of X or Y.

To avoid this break in privacy, we use Beaver triples [7] to mask information about X and Y which allows us to complete the multiplication securely. Beaver triples are defined as three randomly generated numbers, U, V, and W. Each are split into random shares such as X and Y; however there is an extra constraint on W such that $W = U \cdot V$. The protocol follows as such:

1. Locally compute $D = X - U$; $E = Y - V$; open D and E

2. Locally compute $Z = W + E \cdot U + D \cdot V + D \cdot E \cdot \hat{b}$

Note that “opening” a vector means that all parties send their shares to one another so that each party can reconstruct the secret and learn the real data. The actual value that was being masked is the opened value. For the case of D above, Alice and Bob would share their values to get $D = D_A + D_B \mod m$.

Example 2. Let $m = 8$. Let $X = 3$, where $X_A = 1$ and $X_B = 2$. Let $Y = 2$, where $Y_A = 0$ and $Y_B = 2$. Let $U = 1$, where $U_A = 1$ and $U_B = 0$. Let $V = 1$, where $V_A = 0$ and $V_B = 1$. Let $W = U \cdot V = 1 \cdot 1 = 1$, where $W_A = 0$ and $W_B = 1$.

1. $$D_A = 1 - 1 = 0$$

   and

   $$D_B = 2 - 0 = 2$$

   $$E_A = 0 - 0 = 0$$

   and

   $$E_B = 2 - 1 = 1$$

   Open D and E

   $$D = 0 + 2 \mod m = 2$$

   and

   $$E = 0 + 1 \mod m$$

2. $$Z_A = 0 + 1 \cdot 1 + 2 \cdot 0 + 2 \cdot 1 \cdot 1 = 3$$

   $$Z_B = 1 + 1 \cdot 0 + 2 \cdot 1 + 2 \cdot 1 \cdot 0 = 3$$

   $$Z = 3 + 3 \mod m = 6$$

   which in fact is $3 \cdot 2$.

3.1.6 Bitwise And

Bitwise "and" (logical conjunction, denoted by &) is just like the multiplication protocol but the Beaver triples satisfy an exclusive "or" operation, denoted by $\oplus$, instead of a sum. The means the integers are shared by satisfying $X = X_0 \oplus X_1 \oplus \ldots \oplus X_k$. W is randomly generated while satisfying $V \& U$ instead of a multiplication of the two. The protocol is as follows:

1. Locally compute $D = X \oplus U$; $E = Y \oplus V$; open D and E
2. Locally compute $Z = W \oplus (E \& U) \oplus (D \& V) \oplus (D \& E \& \hat{b})$

Example 3. Let $m = 8$. Let $X = 3$, where $[\![X_A]\!]_m = 4$ and $X_B = 7$. Let $Y = 2$, where $Y_A = 0$ and $Y_B = 2$. Let $U = 1$, where $U_A = 1$ and $U_B = 0$. Let $V = 1$, where $V_A = 0$ and $V_B = 1$. Let $W = U \& V = 1 \& 1 = 1$, where $W_A = 0$ and $W_B = 1$.

1.
$$D_A = 4 \oplus 1 = 5$$
and
$$D_B = 7 \oplus 0 = 7$$
$$E_A = 0 \oplus 0 = 0$$
and
$$E_B = 2 \oplus 1 = 3$$
Open D and E
$$D = 5 \oplus 7 \mod m = 2$$
and
$$E = 0 \oplus 3 \mod m = 3$$

2.

$$Z_A = 0 \oplus (3\&1) \oplus (2\&0) \oplus (2\&3\&1) = 0$$

$$Z_B = 1 \oplus (3\&0) \oplus (2\&1) \oplus (2\&3\&0) = 1$$

$$Z = 3\&2 \mod m = 0$$

which in fact is 3&2.

3.1.7 Trusted Initializer

In our computations, we use what is known as a Trusted Initializer (TI). This TI will generate all of the necessary Beaver triples in advance in order to speed up total running time. When ready to participate in the computations, Alice and Bob will first connect to the TI and the TI will send over each party's shares of the triples. Once all shares are received, the parties can participate in the computations and the TI disconnects from the parties and is no longer a part of the protocols.

3.2 Basic Protocols

3.2.1 Batch Multiplication

In a single multiplication, we compute and open D and E. This means for every multiplication, there needs to be a communication between the parties in order to open D and E. The only difference in batch multiplication is we locally compute a batch size, b, worth of D and E, where b is defined specifically for the needs of a specific model being run. Then in a single communication, the parties open b shares of D and E. This reduces the required connections from b times to 1, trading the number of times connecting to a socket for payload size. Each socket connection takes time to establish the connection which means we save time by performing a batch multiplication. The input to this protocol is two vectors, an X and a Y vector, of numbers to be multiplied where $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots, y_n)$, where the X and Y vectors are split as secret shares among the parties. X and Y are multiplied in a pairwise manner to return a vector of secret shares of $(x_1 \cdot y_1, x_2 \cdot y_2, \ldots, x_n \cdot y_n)$.

3.2.2 Batch Bitwise And

Similar to Batch Multiply, we compute multiple Bitwise And operations in a single batch and send them over in one communication. This saves time and allows us to work with vectors of integers at a time.

3.2.3 Parallel Multiplication

Parallel multiplication takes in a vector of numbers and multiplies them pairwise until there is a single number left. In other words, we take a vector $(x_1, x_2, \ldots, x_n)$ and multiply it together resulting in $x_1 \cdot x_2 \cdot \ldots \cdot x_n$. Our specific implementation splits the vector in half and applies batch multiplication on the two halves repeatedly to get the result. This means we take $(x_1, x_2, \ldots, x_n)$ and split it into $(x_1, x_2, \ldots, x_{n/2})$ and $(x_{(n/2+1)}, \ldots, x_n)$ as the x and y vectors of batch multiplication respectively. The run time of this is $\Theta(\log(n))$ as we parallelized each iteration through batch multiplication and each iteration cuts the remaining number of elements to be multiplied in half.

3.2.4 Dot Product

The dot product will return the secret shared dot product of two secret shared vectors, X =$(x_1, x_2, \ldots, x_n)$ and Y =$(y_1, y_2, \ldots, y_n)$. The result is secret shares of $x_1 \cdot y_1 + x_2 \cdot y_2 + \ldots + x_n \cdot y_n$.

1. Jointly compute $Z = BatchMultiply(X, Y)$
2. Locally compute the shares of the result: $z = \sum_{i=1}^{n} [\![Z_i]\!]_m$

Example 1. Let vector $X = (3, 2, 7)$ and vector $Y = (1, 3, 2)$ then $X \cdot Y = 3 \cdot 1 + 2 \cdot 3 + 7 \cdot 2 = 3 + 6 + 14 = 23$.

1. Let Alice's shares be $X_A = (1, 0, 6)$ and $Y_A = (0, 2, 1)$. Let Bob's shares be $X_B = (2, 2, 1)$ and $Y_B = (1, 1, 1)$. Say that after the batch multiplication, Alice's shares are $Z_A = (1, 4, 10)$ and Bob's shares are $Z_B = (2, 2, 4)$.

2. Alice and Bob locally sum up their shares to get $z_A = 1 + 4 + 10 = 15$ and $z_B = 2 + 2 + 4 = 8$.

3.2.5 Exclusive Or

The xor protocol takes two lists of secret shared integers $[X]$ and $[Y]$ and returns the xor of the two numbers. The protocol is as follows:

1. Jointly compute $Z = BatchMultiply(X, Y)$

2. Locally compute the share of the result: $X + Y - (2 \cdot Z)$

Example 2. To make the example easier to understand, we will do the xor on a single integer rather than a list, but the following operations would be done on every element of the inputted integer lists. All operations are done over integer values. The binary representations are included in the example purely for the purpose of illustrations. Let $X = 2$ and $Y = 3$, then the bits of X are $(0, 1, 0)$ and the bits of Y are $(0, 1, 1)$ which means the xor should be shares of $(0, 0, 1)$.

Let $X_A = 1$, $X_B = 1$, $Y_A = 0$ and $Y_B = 3$. Let $Z = X \cdot Y = 6$.

1. After batch multiplication, let $Z_A = 2$ and $Z_B = 4$.

2. Alice locally computes $1 + 0 - (2 \cdot 2) = -3 = (1, 0, 1)$ and Bob locally computes $3 + 1 - (2 \cdot 4) = -4 = (1, 0, 0)$. $(1, 0, 1)$ xor $(1, 0, 0)$ are additive shares of $(0, 0, 1)$

Chapter 4

PRIVACY PRESERVING TEXT CLASSIFICATION

4.1 Overview

Our general protocol for privacy-preserving (PP) text classification relies on several building blocks that are used together to accomplish Step 1 in Fig. 1.1: 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 Logistic Regression (LR) or AdaBoost classifier that is trained on unigrams and bigrams as features, as explained in Chapter 2. 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 machine learning (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 [42], 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 [11].[1] 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.

[1]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 in Chapter 5, $p = 1,301,081$, $a = 972$, and $b = 52,097$.

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 chapter, 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.

4.2 Cryptographic Protocols

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 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 $A = \{a_1, a_2, \ldots, a_m\}$ and Bob has a set $B = \{b_1, b_2, \ldots, b_n\}$. 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

$$x_i = \begin{cases} 1 & \text{if } b_i \in A \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Protocol π_{FE}:

- Alice and Bob secret share each a_j $(j = 1, \ldots, m)$ and each b_i $(i = 1, \ldots, n)$ with each other.

- For $i = 1 \ldots n$: // Computation of secret shares of x_i as defined in Equation (4.1).

    For $j = 1 \ldots m$:

        Alice and Bob run the secure equality test protocol π_{EQ} to compute secret shares

$$x_{ij} = 1 \text{ if } a_j = b_i; \quad x_{ij} = 0 \text{ otherwise} \tag{4.2}$$

    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 [12, 28]. Our solution π_{FE} is tailored to be used within the privacy-preserving machine learning (PPML) framework presented in Chapter 5 (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 [43] 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. [32] 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 (dot 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. To this end, we use <u>protocol π_{decomp}:</u>

$$a_i = \begin{cases} x_i & \text{if } \hat{b} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

$$b_i = \begin{cases} 0 & \text{if } \hat{b} = 1 \\ x_i & \text{otherwise} \end{cases} \tag{4.4}$$

$[\![c_1]\!] \leftarrow Multiply(a_1, b_1)$

$[\![y_i]\!] = [\![x_i]\!]$

For $i \leftarrow 2 \cdots \ell$

```
$[\![d_i]\!] \leftarrow Multiply(a_i, b_i) + 1$
$[\![e_i]\!] \leftarrow Multiply(y_i, c_{i-1}) + 1$
$[\![c_i]\!] \leftarrow Multiply(e_i, d_i) + 1$
$[\![x_i]\!] \leftarrow [\![y_i]\!] + [\![c_{i-1}]\!]$
END
return $[\![x_i]\!]$
```

- A protocol for secure conversion from $\mathbb{Z}_2$ to $\mathbb{Z}_q$: Alice and Bob have as 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 $\mathcal{M}(x)$. Protocol π_{LR} for secure classification with LR models is a follows:

- Input secret shared binary feature vector $(x_0, \cdots, x_n)$, model thresholds $(y_0, \cdots, y_n)$, and an intercept b

- Let xor be the converted the feature vector using π_{2toQ}

- $[\![d]\!] = DotProduct(y, xor) + b$

- Let decomp = $\pi_{\mathsf{decomp}}([\![d]\!])$
- Let zero = the a bit string of the integer $2^m - 1$
- return $\pi_{\mathsf{DC}}(decomp, zero)$

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 [31], 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. 4.1).
- 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.

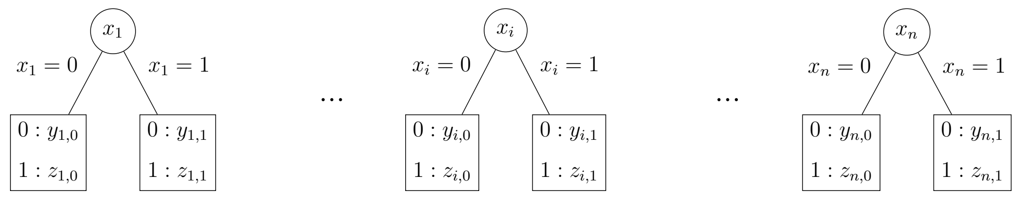

Figure 4.1: Ensemble of decision stumps. Each root corresponds to a feature x_i. The leaves contain weights $y_{i,k}$ for the votes for class label 0 and weights $z_{i,k}$ for the votes for class label 1.

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 opened 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.

4.2.1 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.

Chapter 5
IMPLEMENTATIONS AND RESULTS

5.1 Lynx in Java

5.1.1 Overview

Lynx[1] is an open source framework for privacy-preserving machine learning (PPML) developed and maintained at UW Tacoma. The original version of Lynx was developed in Java. It uses a command line interface and is organized around three different entities. These entities are the broadcast agent, trusted initializer, and the computing parties.

The *broadcast agent* is the intermediate entity to receive and broadcast the communications to the correct computing parties. It is used to reduce the communication cost from $n(n-1)$ to $2n$. If there are n parties, there will be two kinds of communications, one to the broadcast agent and one from the broadcast agent to the destination ($2n$), instead of the $n(n-1)$ communications which would result from every computing party having to communicate to every other party participating in the computations. This organization around a broadcast agent is very meaningful in the setting in which it was first developed, namely to facilitate SMC with a high number of computing parties [1]. Our use case, secure text classification, inherently involves only two parties, namely the party who has the text to be classified (Alice) and the party with the machine learning model that can do the classification (Bob).

The Java-Lynx framework also uses a *trusted initializer (TI)* which generates all of the Beaver triples needed for the executions of the secure multiplication protocol. The parties connect to the initializer to receive their shares of the correlated randomness (the Beaver

[1]https://bitbucket.org/uwtppml/lynx

triples). Once all such shares are distributed, the initializer's job is done and it goes offline.

The final entity are the *computing parties*. These entities are the parties that execute the protocols and use their resources to perform the computations. These are the workforce of the framework.

5.1.2 Secure Text Classification in Java-Lynx

The protocols used in this thesis are listed below, with a reference to their implementations in Java-Lynx. Byte, Integer and Real correspond to what the inputs represent, integer being integer values, byte being binary values and real being a decimal number converted to an integer with finite precision.

- Multiplication: MultiplicationByte.java, MultiplicationInteger.java, MultiplicationReal.java
- Batch Multiplication: BatchMultiplicationByte.java, BatchMultiplicationInteger.java, BatchMultiplicationReal.java
- Parallel Multiplication: ParallelMultiplication.java
- Dot Product: DotProduct.java
- Exclusive Or: OR_XOR.java
- Secure Equality Test π_{EQ}: EqualityByte.java
- Secure Feature Vector Extraction π_{FE}: PrivateSetIntersection.java
- Secure Comparison Test π_{DC}: Comparison.java
- Secure Bit Decomposition π_{decomp}: BitDecomposition.java
- Secure Logistic Regression Classification π_{LR}: LogisticRegressionScoring.java
- Secure AdaBoost Classification π_{AB}: BinaryAdaboostScoring.java

5.2 Lynx in Rust

5.2.1 Overview

Rust-Lynx uses the same architecture as Java-Lynx with the exception of the broadcast agent. We opted to use a 2-party system instead of n parties. Since both versions used two parties, we can figure out the communication cost of the network. With the broadcast agent, we had $2n = 2(2) = 4$ communications in order to transfer a shares during the multiplication protocol. Without the broadcast agent, we have $n(n-1) = 2(2-1) = 2$ communications. This means for our specific feature extraction protocols using 2 parties, we significantly reduce the communication costs by eliminating the broadcast agent.

Another optimization comes from the language of Rust. Rust is a lower level language which means it runs closer to the hardware and is faster by nature. Additionally, Java is object oriented which means we use classes and make function calls on the classes which result in a memory location being called and then another lookup for the function. Our Rust implementation, on the other hand, uses function and eliminates the step of storing and retrieving the class information.

In Lynx computations are done using a modulus, m. This modulus is used as the basis of the additive secret sharing. m can be any integer which we exploit using Rust. In the Rust framework, we use the Wrapping class on the primitive integers to allow the free computation of the modulus. We pick m to be the bit size of the processor. This means for a 64 bit processor, we choose m to be 64 and then instead of the computation, Rust allows the bits to overflow and disregard them, essentially removing the calculation of the modulus.

5.2.2 Secure Text Classification in Rust-Lynx

Below are the protocols added to the Rust-Lynx repository:

- Batch Multiplication: batch_multiply
- Parallel Multiplication: parallel_mult

- Dot Product: dot_product
- Exclusive Or: xor
- Secure Equality Test π_{EQ}: equality_byte
- Secure Feature Vector Extraction π_{FE}: private_set_intersection
- Bitwise And: batch_bitwise_and
- Secure Comparison Test π_{DC}: compare
- Secure Bit Decomposition π_{decomp}: batch_bit_decomp
- Secure Logistic Regression Classification π_{LR}: logistic_regression
- Secure AdaBoost Classification π_{AB}: binary_ada_boost

5.3 Results

We evaluate the proposed protocols in a use case for the detection of hate speech in short text messages, using data from [6]. As explained in Chapter 2, 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 93,629distinct 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). 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 privacy-preserving text classification based on SMC is feasible in practice.

For both our implementations in Java-Lynx and Rust-Lynx, we ran experiments on AWS c5.9xlarge machines with 36 vCPUs, 72.0 GiB Memory. [2] Each of the parties ran on separate machines (connected with a Gigabit Ethernet network), which means that the results in Table 5.1 and 5.2 cover communication time in addition to computation time. Each runtime experiment was repeated 3 times and average results are reported. In Table 5.1 and 5.2 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).

Our results showed that for all hyper-parameter choices of the models, our Rust-Lynx improved the runtimes over Java-Lynx except for the case of logistic regression using unigrams and bigrams with 500 features where we tied the Java results. However, it is important to note that while the scoring times were significantly faster in Rust-Lynx, the feature extraction was actually slower. This is due to the fact that we made a decision to transfer all batch multiplications on a single thread so we wouldn't have to deal with syncing party communications over different threads. The Java-Lynx library had an executor service that managed this which was not available in our current build of Rust. Hence, the faster feature extraction in Java-Lynx came from the fact that the batch multiplication was parallelized using threads and the feature extraction is dominated by comparisons, which relies heavily on multiplications.

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

Table 5.1: 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 **Java framework**, broken down in time needed for feature vector extraction (Extr) and time for feature vector classification (Class).

Java-Lynx	**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

Table 5.2: 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 **Rust framework**, broken down in time needed for feature vector extraction (Extr) and time for feature vector classification (Class).

Rust-Lynx	**Unigrams**				**Unigrams+Bigrams**			
	Acc	**Time (in sec)**			**Acc**	**Time (in sec)**		
		Extr	Class	Tot		Extr	Class	Tot
Ada; 50 trees; depth 1	71.6%	0.925	0.062	0.987	73.3%	2.583	0.064	2.647
Ada; 200 trees; depth 1	73.0%	3.652	0.062	3.714	74.2%	9.915	0.062	9.977
Ada; 500 trees; depth 1	73.9%	9.227	0.066	9.293	74.4%	25.685	0.066	25.751
Logistic regression (50 feat.)	72.4%	0.915	0.038	0.953	73.8%	2.583	0.041	2.624
Logistic regression (200 feat.)	73.3%	3.652	0.039	3.691	73.7%	9.915	0.042	9.957
Logistic regression (500 feat.)	73.4%	9.227	0.041	9.268	74.2%	25.685	0.041	25.726

Chapter 6
CONCLUSIONS

In this thesis we have presented the first provably secure method for privacy-preserving (PP) classification of unstructured text. The appendix contains and analysis of the correctness and security of our solution. As a side result, we have also presented a novel protocol for binary classification over binary input features with an ensemble of decisions stumps.

Two implementations of the protocols in Java and Rust, 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.

BIBLIOGRAPHY

[1] Anisha Agarwal, Rafael Dowsley, Nicholas D McKinney, Dongrui Wu, Chin-Teng Lin, Martine De Cock, and Anderson CA Nascimento. Protecting privacy of users in brain-computer interface applications. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(8):1546–1555, 2019.

[2] 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.

[3] 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.

[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. In *STOC*, volume 97, pages 446–455, 1997.

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

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

[10] 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.

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

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

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

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] Rafael Baião Dowsley. *Cryptography Based on Correlated Data: Foundations and Practice Cryptography Based on Correlated Data: Foundations and Practice*. PhD thesis, Karlsruhle Institute of Technology, 2016.

[28] 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`.

[29] 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.

[30] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *ICML*, volume 96, pages 148–156, 1996.

[31] 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.

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

[33] 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.

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

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

[36] 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.

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

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

[39] Fei Li and Hong Yu. ICD coding from clinical text using multi-filter residual convolutional neural network. In *Proceedings of the 34th AAAI Conference on Artificial Intelligence*, 2020.

[40] 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.

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

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

[43] 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.

[44] 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.

[45] Devin Reich, Ariel Todoki, Rafael Dowsley, Martine De Cock, and Anderson Nascimento. Privacy-preserving classification of personal text messages with secure multi-party computation. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3752–3764. Curran Associates, Inc., 2019.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

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 [8] 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. [15] 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.1.1 ([8]) *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. [15].

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 [9, 10]. Multiple setup assumptions were used previously to achieve UC-security for secure computation protocols, including: the availability of a common reference string [9, 10, 41], the availability of a public-key infrastructure [4], the random oracle model [34, 5], the existence of noisy channels between the parties [22, 26], and the availability of signature cards [35] or tamper-proof hardware [37, 21, 23]. 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

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

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

Figure A.1: The Trusted Initializer Functionality.

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., [46, 25, 24, 36, 48, 19, 16, 17, 31, 18]. The trusted initializer is modeled by the ideal functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ described in Figure A.1.

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 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.

A.2 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 A.2 [27, 18]. The protocol for secure comparison π_{DC} UC-realizes the functionality $\mathcal{F}_{\mathsf{DC}}$ described in Figure A.3 [32, 18]. The protocol for secure bit-decomposition π_{decomp} UC-realizes the functionality $\mathcal{F}_{\mathsf{decomp}}$ described in Figure A.4 [18]. The LR classification protocol π_{LR} UC-realizes the functionality $\mathcal{F}_{\mathsf{LR}}$ described in Figure A.5 [18].

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 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 $\mathcal{S}$

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 A.2: The Distributed Matrix Multiplication Functionality.

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 A.3: 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 A.4: 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 A.5: The Logistic Regression Classification Functionality.

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 $z = x_A + x_B - 2x_Ax_B$ 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 $\mathcal{F}_{\mathsf{AB}}$.

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 A.6: The Equality Test Functionality.

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 A.7: 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 A.8: 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 A.9: The AdaBoost Classification Functionality.

A.3 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 $\mathcal{F}_{\mathsf{FE}}$, $\mathcal{F}_{\mathsf{2toQ}}$ 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 $\mathcal{F}_{\mathsf{TC-AB}}$.

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 $\mathsf{sign}\left(\langle x, w\rangle + b\right)$ and output it to Bob as the class prediction.

Figure A.10: 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 A.11: The Functionality for Privacy-Preserving Text Classification with Adaboost.