Fast Privacy-Preserving Text Classification based on Secure Multiparty Computation

Amanda Resende, Davis Railsback, Rafael Dowsley, Anderson C. A. Nascimento and Diego F. Aranha

Abstract—We propose a privacy-preserving Naive Bayes classifier and apply it to the problem of private text classification. In this setting, a party (Alice) holds a text message, while another party (Bob) holds a classifier. At the end of the protocol, Alice will only learn the result of the classifier applied to her text input and Bob learns nothing. Our solution is based on Secure Multiparty Computation (SMC). Our Rust implementation provides a fast and secure solution for the classification of unstructured text. Applying our solution to the case of spam detection (the solution is generic, and can be used in any other scenario in which the Naive Bayes classifier can be employed), we can classify an SMS as spam or ham in less than 340 ms in the case where the dictionary size of Bob's model includes all words ($n = 5200$) and Alice's SMS has at most $m = 160$ unigrams. In the case with $n = 369$ and $m = 8$ (the average of a spam SMS in the database), our solution takes only 21 ms.

Index Terms—Privacy-Preserving Classification, Secure Multiparty Computation, Naive Bayes, Spam.

I. Introduction

Classification is a supervised learning technique in Machine Learning (ML) that has the goal of constructing a classifier given a set of training data with class labels. Decision Tree, Naive Bayes, Random Forest, Logistic Regression and Support Vector Machines (SVM) are some examples of classification algorithms. These algorithms can be used to solve many problems, such as: classifying an email/Short Message Service (SMS) as spam or ham (not spam) [5]; diagnosis of a medical condition (disease versus no disease) [60]; hate speech detection [52]; face classification [43]; fingerprinting identification [15]; and image categorization [27]. For the first three examples above, classification is binary, where there are only two class labels (yes or no); while the last three are multi-class, that is, there are more than two classes.

We consider the scenario in which there are two parties: one possesses the private data to be classified and the other party holds a private model used to classify such data. In such a scenario, the party holding the data (Alice) is interested in obtaining the classification result of such data against a model held by a second party (Bob) so that, at the end of the classification protocol, Alice knows solely the input data and the classification result, and Bob knows nothing beyond

A. Resende is with the Institute of Computing, University of Campinas, Campinas, Brazil. E-mail: amanda.resende@ic.unicamp.br.

D. Railsback and A. C. A. Nascimento are with the School of Engineering and Technology, University of Washington Tacoma, WA 98402, USA. E-mails: {drail, andclay}@uw.edu.

R. Dowsley is with the Faculty of Information Technology, Monash University, Australia. E-mail: rafael.dowsley@monash.edu.

D. F. Aranha is with the Department of Computer Science, Aarhus University, Aarhus, Denmark. E-mail: dfaranha@cs.au.dk.

the model itself. This scenario is a very relevant one. There are many situations where a data owner is not comfortable sharing a piece of data that needs classification (think of psychological or health related data). Also, a machine learning model holder might not want to/cannot reveal the model in the clear for intellectual property issues or because the model reveals information about the data set used to train it. Thus, both parties have proper incentives to participate in a protocol providing the joint functionality of private classification.

Due to these concerns, mechanisms such as Secure Multiparty Computation (MPC) [18], Differential Privacy (DP) and Homomorphic Encryption (HE) can be used to build privacy-preserving solutions. MPC allows two or more parties to jointly compute a function over their private inputs without revealing any information to the other party, whereas HE is an encryption scheme that allows performing computations on encrypted data without having to decrypt it. And, DP is a technique that adds random noise to queries, to prevent an adversary from learning information about any particular individual in the data set.

Our main goal is to propose protocols for privacy-preserving text classification. By carefully selecting engineering optimizations, we improve upon previous results by Reich *et al.* [52] by over one order of magnitude achieving, to the best of our knowledge, the fastest text-classification results in the available literature (21ms for an average sample of our data set). More specifically, we propose a privacy-preserving Naive Bayes classification (PPNBC) based on MPC where given a trained model we classify/predict an example without revealing any additional information to the parties other than the classification result, which can be revealed to one specified party or both parties. We then apply our solution to a text classification problem: classifying SMSes as spam or ham.

A. Application to Private SMS Spam Detection

SMS is one of the most used telecommunication service in the world. It allows mobile phone users to send and receive a short text (which has 160 7-bit characters maximum). Due to advantages such as reliability (since the message reaches the mobile phone user), low cost to send an SMS (especially if bought in bulk), the possibility of personalizing, and immediate delivery, SMS is a widely used communication medium for commercial purposes, and mobile phone users are flooded with unsolicited advertising.

SMSes are also used in scams, where someone tries to steal personal information, such as credit card details, bank account information, or social security numbers. Usually, the scammer

sends an SMS with a link that invites a person to verify his/her account details, make a payment, or that claims that he/she has earned some amount of money and needs to use the link to confirm. In all cases, such SMSes can be classified as spam.

Machine learning classifiers can be used to detect whether an SMS is a spam or not (ham). During the training phase, these algorithms learn a model from a data set of labeled examples, and later on, are used during the classification/prediction phase to classify unseen SMSes. In a Naive Bayes classifier, the model is based on the frequency that each word occurs in the training data set. In the classification phase, based on these frequencies, the model predicts whether an unseen SMS is spam or not.

A concern with this approach is related to Alice's privacy since she needs to make her SMSes available to the spam filtering service provider, Bob, which owns the model. SMSes may contain sensitive information that the user would not like to share with the service provider. Besides, the service provider also does not want to reveal what parameters the model uses (in Naive Bayes, the words and its frequencies) to spammers and concurrent service providers. Our privacy-preserving Naive Bayes classification (PPNBC) based on MPC, provides an extremely fast secure solution for both parties to classify SMSes as spam or ham without leaking any additional information while maintaining essentially the same accuracy as the original algorithm performed in the clear. While our experimental treatment is focused on SMS messages, the same approach can be naturally generalized to classify short messages received over Twitter or instant messengers such as WhatsApp or Signal.

B. Our Contributions

These are the main contributions of this work:

- **A privacy-preserving Naive Bayes classification (PPNBC) protocol**: We propose the first privacy-preserving Naive Bayes classifier with private feature extraction. Previous works assumed the features to be publicly known. It is based on secret sharing techniques from MPC. In our solution, given a trained model it is possible to classify/predict an example without revealing any additional information to the parties other than the classification result, which can be revealed to one or both parties. We provide a security proof for the proposed protocol using the Universal Composability (UC) framework [12], thus proving that it enjoys strong securities guarantees and can be arbitrary composed without compromising the security.
- **An efficient and optimized software implementation of the protocol:** The proposed protocol is implemented in Rust using an up-to-date version of the RustLynx framework available at https://bitbucket.org/uwtppml/rustlynx/src/master/.
- **Experimental results for the case of SMS classification as spam/ham**: The proposed protocol is evaluated in a use case for SMS spam detection, using a data set widely used in the literature. However, it is important to note that the solution is generic, and can be used in

any other scenario in which the Naive Bayes classifier can be employed.

While the necessary building blocks already exist in the literature, the main novelty of our work is putting these building blocks together, optimizing their implementations and obtaining the fastest protocol for private text classification to date.

C. Organization

This paper is organized as follows. In Section 2, we define the notation used during this work, describe the necessary cryptographic building blocks and briefly introduce the Naive Bayes classifier. In Sections 3 and 4, we describe our Privacy-Preserving Naive Bayes Classification (PPNBC) protocol and present its security analysis. In Section 5, we describe the experimental results, from the training phase until the classification using our PPBNC, as well as the cryptographic engineering techniques used in our implementation. Finally, in Sections 6 and 7, we present the related works and the conclusions.

II. PRELIMINARIES

As in most existing works on privacy-preserving machine learning based on MPC, we consider an honest-but-curious adversary (also known as semi-honest adversary). In this model, each party follows the protocol specifications but may try to learn from his view of the protocol execution additional information other than his input and specified output.

A. Secure Computation Based on Additive Secret Sharing

Our solution is based on additive secret sharing over a ring $\mathbb{Z}_q = \{0, 1, 2, \cdots, q-1\}$. A value x is secret shared between Alice and Bob over $\mathbb{Z}_q$ by picking $x_A, x_B \in \mathbb{Z}_q$ uniformly at random with the constraint that $x = x_A + x_B \mod q$. Alice receives the share x_A while Bob receives the share x_B. We denote this pair of shares by $[\![x]\!]_q$. A secret shared value x can be open towards one party by disclosing both shares to that party. Let $[\![x]\!]_q$, $[\![y]\!]_q$ be secret shared values and c be a constant. Alice and Bob can perform locally and straightforwardly the following operations:

- Addition ($z = x + y$): Each party locally adds its local shares of x and y modulo q in order to obtain a share of z. This operation will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + [\![y]\!]_q$.
- Subtraction ($z = x - y$): Each party locally subtracts its local share of x and y modulo q in order to obtain a share of z. This operation 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 modulo q to obtain a share of z. This operation 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 $z_B = x_B$, keeping his original share. This will be denoted by $[\![z]\!]_q \leftarrow [\![x]\!]_q + c$.

Unlike the above operations, secure multiplication of secret shared values (i.e., $z = xy$) cannot be done locally and

requires interaction between Alice and Bob. To compute this operation highly efficiently, we use Beaver's multiplication triple technique [9], which consumes a multiplication triple $(\llbracket u \rrbracket_q, \llbracket v \rrbracket_q, \llbracket w \rrbracket_q)$ such that $w = uv$ in order to compute the multiplication of $\llbracket x \rrbracket_q$ and $\llbracket y \rrbracket_q$ without leaking any information. We use a trusted initializer (TI) to generate uniformly random multiplication triples and secret share them to Alice and Bob. In the trusted initializer model, TI can pre-distribute correlated randomness to Alice and Bob during a setup phase, which is run before the protocol execution (possibly long before Alice and Bob get to know their inputs). The TI is not involved in any other part of the protocol execution and does not get to know the parties inputs and outputs.[1] This model was used in many previous works, e.g., [55], [33], [32], [39], [57], [21], [22], [35], [2]. If a TI is not desirable or unavailable, Alice and Bob can securely simulate a TI at the cost of introducing computational assumptions in the protocol [23]. The TI is modeled by the ideal functionality $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$. The TI additionally generates random values in $\mathbb{Z}_q$ and delivers them to Alice so that she can use them to secret share her inputs. If Alice wants to secret share an input x, she picks an unused random value r (note that Bob does not know r), and sends $c = x - r$ to Bob. Her share x_A of x is then set to $x_A = r$, while Bob's share x_B is set to $x_B = c$. The secret sharing of Bob's inputs is done similarly using random values that the TI only delivers to him.

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

The following straightforward extension of Beaver's idea performs the UC-secure multiplication of secret shared matrices $X \in \mathbb{Z}_q^{i \times j}$ and $Y \in \mathbb{Z}_q^{j \times k}$ [29], [23]. The protocol will be denoted by π_{DMM} and works as follows:

1) The TI 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 $\llbracket U \rrbracket_q, \llbracket V \rrbracket_q, \llbracket W \rrbracket_q$ (the secret sharings are done element-wise) to Alice and Bob.
2) Alice and Bob locally compute $\llbracket D \rrbracket_q \leftarrow \llbracket X \rrbracket_q - \llbracket U \rrbracket_q$ and $\llbracket E \rrbracket_q \leftarrow \llbracket Y \rrbracket_q - \llbracket V \rrbracket_q$, and then open D and E.
3) Alice and Bob locally compute $\llbracket Z \rrbracket_q \leftarrow \llbracket W \rrbracket_q + E\llbracket U \rrbracket_q + D\llbracket V \rrbracket_q + DE$.

B. Fixed Point Arithmetic

Many real-world applications of MPC require representing and operating on continuous data. This poses a challenge because the security of additive secret sharing depends on the fact that shares are uniformly random – a concept that only

---

[1]It is a well-known fact that UC-secure MPC needs a setup assumption [13], [14]. A TI is one of the setup assumptions that allows obtaining UC-secure MPC. Other setup assumptions that enable UC-secure MPC include: a common reference string [13], [14], [49], the availability of a public-key infrastructure [6], signature cards [38], tamper-proof hardware [41], [28], [31] or noisy channels between the parties [30], [34], and the random oracle model [37], [8], [20].

exists for samples of finite sets. For compatibility with MPC, continuous values need to be represented within a range of possible values. We use the mapping

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

to represent real numbers in $\mathbb{Z}_q$, where $q = 2^\lambda$, as fixed-point precision two's complement values. The parameter a is the *fractional accuracy* – the number of bits used to represent negative powers of 2. This mapping preserves addition in $\mathbb{Z}_q$ straightforwardly, but a multiplication of two fixed-point values results in a fixed point value with $2a$ fractional bits. To maintain the expected representation in $\mathbb{Z}_q$, all products need to be truncated by a bit positions, requiring an additional MPC protocol [48]. In this paper, fixed-point values are only added together and multiplied by 0 or 1, so a truncation protocol is not needed for our purposes.

C. Cryptographic Building Blocks

Next, we present the cryptographic building blocks that are used in our PPNBC solution.

Secure Equality Test: To perform a secure equality test, we use a straightforward folklore protocol π_{EQ}. As input, Alice and Bob have bitwise secret sharings in $\mathbb{Z}_2$ of the bitstrings $X = \{x_\ell, \ldots, x_1\}$ and $Y = \{y_\ell, \ldots, y_1\}$. The protocol generates as output a secret sharing of 1 if $X = Y$ and a secret sharing of 0 otherwise. The protocol π_{EQ} works as follows:

1) For $i = 1, \ldots, \ell$, Alice and Bob locally compute $\llbracket r_i \rrbracket_2 \leftarrow \llbracket x_i \rrbracket_2 + \llbracket y_i \rrbracket_2 + 1$.
2) Alice and Bob use secure multiplication (π_{DMM}) to compute a secret sharing of $z = r_1 \cdot \ldots \cdot r_\ell$. They output the secret sharing $\llbracket z \rrbracket_2$. (Note that if $x = y$, then $r_i = 1$ in all i positions, thus $z = 1$; otherwise some $r_i = 0$ and so $z = 0$).

By performing the multiplications to compute z in a tree style with the values $r_1, \ldots, r_\ell$ in the ℓ leaves, the protocol π_{EQ} requires $\lceil \log(\ell) \rceil$ rounds of communication and a total of $\sum_{i=1}^{\lceil \log(\ell) \rceil} \lceil \ell/2^{i-1} \rceil$ bits of data transfer. For batched inputs $\{X_1, \ldots, X_k\}$, $\{Y_1, \ldots, Y_k\}$, the number of communication rounds remains the same and the data transfer per round is scaled by k.

Secure Feature Extraction: To perform the feature extraction in a privacy-preserving way, we use the protocol π_{FE} from Reich et al. [52]. Alice has as input the set $A = \{a_1, \cdots, a_m\}$ of unigrams occurring in her message and Bob has as input the set $B = \{b_1, \cdots, b_n\}$ of unigrams that occur in his ML model. The elements of both sets are represented as bitstrings of size ℓ. The purpose of the protocol is to extract which words from Alice's message appear in Bob's set. Thus, at the end of the protocol, Alice and Bob have secret shares of a binary feature vector Y which represents what words in Bob's set appear in Alice's message. The binary feature vector $Y = \{y_1, \cdots, y_n\}$ is defined as:

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

The protocol π_{FE} works as follows:

1) Alice secret shares a_p with Bob for $p = 1, \cdots, m$, while Bob secret shares b_i with Alice for $i = 1, \cdots, n$. Both use bitwise secret sharings in $\mathbb{Z}_2$. To secret share their input a_p and b_i, Alice and Bob use the method described in Section II-A.

2) For each a_p and each b_i, they execute the secure equality protocol π_{EQ}, which outputs a secret sharing of

$$y'_{ip} = \begin{cases} 1, & \text{if } a_p = b_i \\ 0, & \text{otherwise} \end{cases}$$

3) Alice and Bob locally compute the secret share $[\![y_i]\!]_2 \leftarrow \sum_{p=1}^m [\![y'_{ip}]\!]_2$

The protocol requires $\lceil \log(\ell) \rceil + 1$ rounds of communication and a total of $\sum_{i=1}^{\lceil \log(\ell) \rceil} \lceil \ell/2^{i-1} \rceil$ bits of data transfer for each call of the π_{EQ}. π_{FE} requires $m \times n$ equality tests. The number of communication rounds remains the same as a single execution, as all the tests can be done in parallel. The data transfer however is scaled by $m \times n$.

Secure Conversion: To perform a secure conversion from a secret sharing in $\mathbb{Z}_2$ to a secret sharing in $\mathbb{Z}_q$, we use the secure conversion protocol π_{2toQ} presented by Reich *et al.* [52]. Alice and Bob have as input a secret sharing $[\![x]\!]_2$ and without learning any information about x, they must get a secret sharing $[\![x]\!]_q$. The protocol π_{2toQ} works as follows:

1) For the input $[\![x]\!]_2$, let $x_A \in \{0, 1\}$ denote Alice's share of x and $x_B \in \{0, 1\}$ denote Bob's share.
2) Define $[\![x_A]\!]_q$ as the shares $(x_A, 0)$ and $[\![x_B]\!]_q$ as the shares $(0, x_B)$.
3) Alice and Bob compute $[\![y]\!]_q \leftarrow [\![x_A]\!]_q [\![x_B]\!]_q$.
4) They output $[\![z]\!]_q \leftarrow [\![x_A]\!]_q + [\![x_B]\!]_q - 2[\![y]\!]_q$.

The protocol π_{2toQ} requires 1 round of communication and a total of 2λ bits of data transfer, where λ is the bit length of q. For batched inputs $\{x_1, ..., x_k\}$, the number of communication rounds remains the same and the data transfer is scaled by k.

Secure Bit Extraction: The secure bit extraction protocol π_{BTX} takes a secret value $[\![x]\!]_q$ and a publicly known bit position α and returns a $\mathbb{Z}_2$-sharing of the α-th bit of x, $[\![(x >> (\alpha - 1)) \wedge 1]\!]_2$. The protocol is based on a reduction of the protocol for full bit decomposition modeled after a matrix representation of the carry look-ahead adder circuit that was presented in [25]. π_{BTX} [1] works as follows:

1) For the secret shared value $[\![x]\!]_q = (x_A, x_B)$ such that $x = x_A + x_B \mod q$, Alice and Bob locally create bitwise sharings of the *propagate signal*

$$p^{(\alpha)} \leftarrow x_A^{(\alpha)} \oplus x_B^{(\alpha)}, \ \ldots, \ p^{(1)} \leftarrow x_A^{(1)} \oplus x_B^{(1)},$$

where $x_A^{(i)}/x_B^{(i)}$ indicates the i-th bit of x_A/x_B.
2) Alice and Bob use the secure multiplication to jointly compute the *generate signal*

$$g^{(\alpha)} \leftarrow x_A^{(\alpha)} \wedge x_B^{(\alpha)}, \ \ldots, \ g^{(1)} \leftarrow x_A^{(1)} \wedge x_B^{(1)}.$$

3) Alice and Bob jointly compute the $(\alpha - 1)$-th *carry bit* $[\![c^{(\alpha-1)}]\!]_2$ as the upper right entry of

$$\prod_{1 \leq i \leq \alpha-1} \begin{bmatrix} [\![p^{(i)}]\!]_2 & [\![g^{(i)}]\!]_2 \\ 0 & 1 \end{bmatrix}$$

.

4) Alice and Bob locally compute $[\![x^{(\alpha)}]\!]_2 \leftarrow [\![p^{(\alpha)}]\!]_2 \oplus [\![c^{(\alpha-1)}]\!]_2$.

The protocol π_{BTX} requires one round of communication and 2α bits of data transfer before the matrix composition phase. The matrix composition phase can be performed by computing pairwise compositions of all matrices in $\lceil \log(\alpha - 1) \rceil$ rounds of communication. The total data transfer per matrix multiplication is four bits. Figure 1 shows an example circuit to compute the matrix composition phase for $\alpha = 17$. For batched inputs $\{x_1, \ldots, x_k\}$, the number of communication rounds remains the same and the total data transfer is scaled by k.

Secure Comparison: To perform a secure comparison of secret shared integers, we use the protocol π_{GEQ} of Adams *et al.* [1]. As input, Alice and Bob hold secret shares in $\mathbb{Z}_q$ of integers x and y such that $|x - y| < 2^{\lambda-1}$ (as integers). Particularly, Alice and Bob can use this protocol with integers x and y in the range $[-2^{\lambda-2}, 2^{\lambda-2} - 1]$ (a negative value u is represented as $2^\lambda - |u|$). The protocol returns a secret share in $\mathbb{Z}_2$ of 1 if $x \geq y$ and of 0 otherwise. The protocol π_{GEQ} works as follows:

1) Alice and Bob locally compute the difference of x and y as $[\![\mathrm{diff}]\!]_q \leftarrow [\![x]\!]_q - [\![y]\!]_q$. Note that if $y > x$, then diff is negative.
2) Alice and Bob extract a $\mathbb{Z}_2$-sharing $[\![MSB]\!]_2$ of the most-significant bit (MSB) of diff using the protocol π_{BTX}.
3) Given that the most-significant bit of a secret shared value in $\mathbb{Z}_q$ is 1 if and only if it is negative, the negation of the most-significant bit, $[\![z]\!]_2 \leftarrow 1 + [\![MSB]\!]_2$, is 1 if and only if $x \geq y$.

The protocol π_{GEQ} requires $\lceil \log(\lambda - 1) + 1 \rceil$ rounds of communication and a total of $4\lambda + 4 \log(\lambda - 1) - 6$ bits of data transfer, where λ is the the bit length of q. For batched inputs $\{x_1, ..., x_k\}, \{y_1, ..., y_k\}$, the number of communication rounds remains the same and the data transfer per round is scaled by k.

D. Naive Bayes Classifiers

Naive Bayes is a statistical classifier based on Bayes' Theorem with an assumption of independence among features/predictors. It assumes that the presence (or absence) of a particular feature in a class is unrelated to the presence (or absence) of any other feature. The Bayes' theorem is used as follows:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)},$$

where: (1) c is the class/category; (2) x is the feature vector of test example; (3) $P(c|x)$ is the posterior probability, i.e., given test example x, what is its probability of belonging to class c; (4) $P(x|c)$ is known as the likelihood, i.e., given a class c, what is the probability of example x belonging to class c; (5) $P(c)$ is the class prior probability; (6) $P(x)$ is the predictor prior probability.

The predictor prior probability $P(x)$ is the normalizing constant so that the $P(c|x)$ does actually fall in the range [0, 1]. In our solution we will be comparing the probabilities
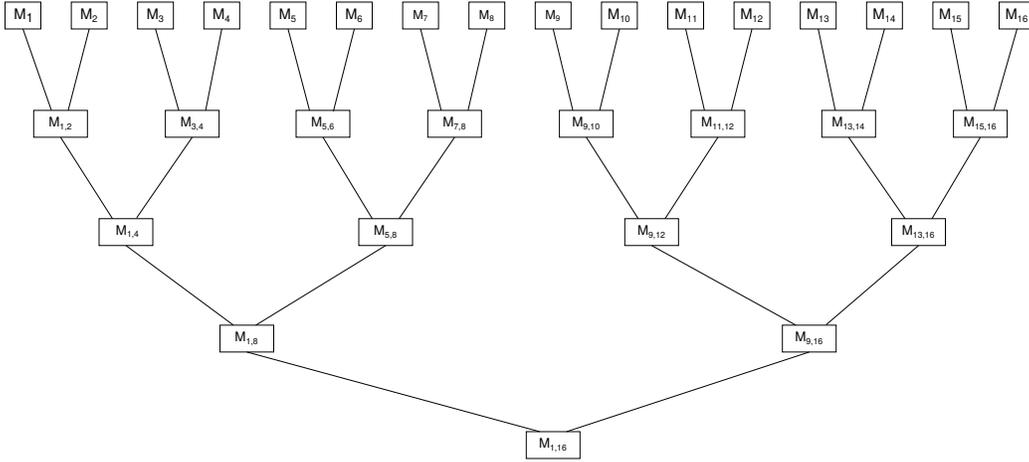
Fig. 1. A circuit to compute the $(\alpha - 1)$-th matrix composition in $\lceil \log(\alpha - 1) \rceil$ layers. The notations $M_{i,j}$ indicates the composition of all matrices from M_i to M_j, inclusive.

of different classes to determine the most likely class of an example. The probabilities are not important per se, only their comparative values are relevant. As the denominator $P(x)$ remains the same, it will be omitted and we will use

$$P(c|x) = P(x|c)P(c).$$

As per assumption the features $x = (x_1, \ldots, x_d)$ are independent, we get

$$P(c|x) = P(c) \prod_{k=1}^{d} P(x_k|c).$$

Note that when executing Naive Bayes, since the probabilities are often very small numbers, multiplying them will result in even smaller numbers, which often results in underflows that can cause the model to fail. To solve that problem and also simplify operations (and consequently improve performance), we will "convert" all multiplication operations into additions by using logarithms. Applying the logarithm we get

$$
\begin{aligned}
\log(P(c|x)) &= \log\left(P(c) \prod_{k=1}^{d} P(x_k|c)\right) \\
&= \log(P(c)) + \sum_{k=1}^{d} \log(P(x_k|c)).
\end{aligned}
$$

To perform the classification, we then compute the argmax

$$\hat{c} = \text{argmax}_c \left[\log(P(c)) + \sum_{k=1}^{d} \log(P(x_k|c))\right],$$

where argmax_c returns the class c that has the highest value for the test example x.

Naive Bayes classifiers differ mainly on the assumptions they make regarding the distribution of $P(x|c)$. In Gaussian Naive Bayes, the assumption is that the continuous values associated with each class are distributed according to a Gaussian distribution. For discrete features, as we have in text classification, we can use the Bernoulli Naive Bayes or the multinomial Naive Bayes. In the Bernoulli Naive Bayes, the features are Boolean variables, where 1 means that the word

occurred in the text and 0 if the word did not occur. And, in the multinomial Naive Bayes, the features are the frequency of the words present in the document. In this work, we use the multinomial Naive Bayes and the frequencies of the words are determined during the training phase.

III. PRIVACY-PRESERVING NAIVE BAYES CLASSIFICATION

For the construction of our Privacy-Preserving Naive Bayes Classification (PPNBC) protocol π_{PPNBC}, Alice constructs her set $A = \{a_1, \ldots, a_m\}$ of unigrams occurring in her message and Bob constructs his set $B = \{b_1, \ldots, b_n\}$ of unigrams that occur in his ML model. Bob also has $\log(P(c_j))$, that is the logarithm of the probability for each class c_j and a set of logarithms of probabilities $\{\log(P(b_1|c_j)), \ldots, \log(P(b_n|c_j)), \log(1 - P(b_1|c_j)), \ldots, \log(1 - P(b_n|c_j))\}$, that is the logarithm of the probability of a word b_i occurring or not in a class c_j. All a_k and b_i are represented as bit strings of length ℓ. In our current implementation, we focus on binary classification. It is straightforward to generalize our protocols to the case of classification into more than two classes by using a secure argmax protocol. Our protocol π_{PPNBC} follows the description of the Naive Bayes presented in Section II-D (using logarithms and not using the normalizing constant), and works as follows:

1) Alice and Bob execute the secure feature extraction protocol π_{FE} with inputs $(a_1, \ldots, a_m)$ and $(b_1, \ldots, b_n)$, respectively. The output consists of secret shared values $[\![y_1]\!]_2, \ldots, [\![y_n]\!]_2$ in $\mathbb{Z}_2$, where $y_i = 1$ if the word $b_i \in A$ and 0 otherwise;

2) They use protocol π_{2toQ} to convert $[\![y_1]\!]_2, \ldots, [\![y_n]\!]_2$ to $[\![y_1]\!]_q, \ldots, [\![y_n]\!]_q$, containing secret sharings of the same values in $\mathbb{Z}_q$;

3) For each class c_j

   a) Using the method described in Section II-A, Bob creates secret shares of his inputs $\log(P(c_j)), \log(P(b_1|c_j)), \ldots, \log(P(b_n|c_j)), \log(1 - P(b_1|c_j)), \ldots, \log(1 - P(b_n|c_j))$, which contain the class probability

and the set of logarithms of the conditional probabilities;

b) For $i = 1, \ldots, n$, Alice and Bob use the protocol π_{DMM} to compute $[\![w_i]\!]_q \leftarrow [\![y_i]\!]_q [\![\log(P(b_i|c_j))]\!]_q + (1 - [\![y_i]\!]_q)[\![\log(1 - P(b_i|c_j))]\!]_q$;

c) Alice and Bob locally compute $[\![u_j]\!]_q \leftarrow [\![\log(P(c_j))]\!]_q + \sum_{i=1}^{n}[\![w_i]\!]_q$.

4) Alice and Bob use the protocol π_{GEQ} to compare the results of Step 3(c) for the two classes, getting as output a secret sharing of the output class $[\![c]\!]_2$ (the secret sharing $[\![c]\!]_2$ can afterwards be opened towards the parties that should receive the result of the classification).

IV. SECURITY

The security model considered in this work is the Universal Composability (UC) framework of Canetti [12], which is the gold standard for formally defining and analyzing the security of cryptographic protocols. Any protocol that is proven UC-secure, can be arbitrarily composed with other copies of itself and of other protocols (even with arbitrarily concurrent executions) while preserving security. That is an extremely useful property that allows the modular design of cryptographic protocols. UC-security is a necessity for cryptographic protocols running in complex environments such as the Internet. Here only a short overview of the UC framework for the specific case of protocols with two participants is presented. We refer interested readers to Cramer et al. [18] for more details.

In the UC framework the security is analyzed by comparing a real world with an ideal world. In the real world Alice and Bob interact between themselves and with an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. The environment $\mathcal{Z}$ captures all external activities to the protocol instance under consideration, and is responsible for giving the inputs and getting the outputs from Alice and Bob. The adversary $\mathcal{A}$ can corrupt either Alice or Bob, in which case he gains the control over that participant. The network scheduling is assumed to be adversarial and thus $\mathcal{A}$ is responsible for delivering the messages between Alice and Bob. In the ideal world, there is an ideal functionality $\mathcal{F}$ that captures the perfect specification of the desired outcome of the computation. $\mathcal{F}$ 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 executed between Alice and Bob in the real world UC-realizes the ideal 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 the participants Alice and Bob, and the adversary $\mathcal{A}$; (2) and an ideal execution with dummy parties (that only forward inputs/outputs), $\mathcal{F}$ and $\mathcal{S}$.

Simplifications: The messages of ideal functionalities are formally public delayed outputs, meaning that $\mathcal{S}$ is first asked whether they should be delivered or not (this is due to the modeling that the adversary controls the network scheduling). This detail as well as the session identifications are omitted from the description of functionalities presented here for the sake of readability.

The protocol π_{DMM} for secure matrix multiplication UC-realizes the distributed matrix multiplication functionality $\mathcal{F}_{\mathsf{DMM}}$ in the trusted initializer model [29], [23].

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

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

As proved by Reich *et al.* [52], the protocol π_{EQ} UC-realizes the functionality $\mathcal{F}_{\mathsf{EQ}}$.

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.

From the fact that π_{EQ} UC-realizes $\mathcal{F}_{\mathsf{EQ}}$, it follows straightforwardly that π_{FE} UC-realizes the functionality $\mathcal{F}_{\mathsf{FE}}$. Note that in an internal simulation of an execution of the protocol π_{FE} for the adversary $\mathcal{A}$, $\mathcal{S}$ can use the leverage of being responsible for simulating $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ in order to extract all inputs of the corrupted party, which can then be forwarded to $\mathcal{F}_{\mathsf{FE}}$.

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.

As proved by Reich *et al.* [52], the protocol π_{2toQ} UC-realizes the functionality $\mathcal{F}_{\mathsf{2toQ}}$.

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.

The bit extraction protocol π_{BTX} of [1] is a straightforward simplification of the bit decomposition protocol $\pi_{\mathsf{decompOPT}}$ from [25] and UC-realizes the bit extraction functionality $\mathcal{F}_{\mathsf{BTX}}$. Note that the simulator $\mathcal{S}$ can trivially extract the bit-string of a corrupted party in an internal simulation of π_{BTX} with the adversary $\mathcal{A}$ by using the fact that it is responsible for simulating $\mathcal{F}_{\mathsf{DMM}}$ that is used to compute the generate signal. Therefore $\mathcal{S}$ has a perfect simulation strategy and $\mathcal{Z}$ cannot distinguish the ideal and real worlds.

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

$\mathcal{F}_{\mathsf{BTX}}$ is parametrized by α. It receives bit-strings $x_A = x_A^{(\alpha)} \cdots x_A^{(1)}$ and $x_B = x_B^{(\alpha)} \cdots x_B^{(1)}$ from Alice and Bob, respectively, and returns a secret sharing of

the α-th bit of $x = x_A + x_B$.

Input: Upon receiving a message from Alice with her bit-string x_A or from Bob with his bit-string x_B, record it, ignore any subsequent messages from that party and inform the other party about the receipt.

Output: Upon receipt of the inputs from both parties, compute $x = x_A + x_B$, extract the α-th bit x_α of x and distribute a new secret sharing $[\![x_\alpha]\!]_2$ of the bit x_α. 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.

The correctness of π_{GEQ} follows trivially. For the simulation, $\mathcal{S}$ executes an internal copy of $\mathcal{A}$ interacting with an instance of protocol π_{GEQ} in which the uncorrupted parties use dummy inputs. Note that all the messages that $\mathcal{A}$ receives look uniformly random to him. Since π_{BTX} is substituted by $\mathcal{F}_{\mathsf{BTX}}$ using the UC composition theorem, and $\mathcal{S}$ is responsible for simulating $\mathcal{F}_{\mathsf{BTX}}$ in the ideal world, $\mathcal{S}$ can leverage this fact in order to extract the share that any corrupted party have of the value diff $= x - y$. Let the extracted value of the corrupted party be denoted by diff_C. The simulator then picks random values x_C, y_C in $\mathbb{Z}_q$ such that $\mathsf{diff}_C = x_C - y_C$ and submit these values to $\mathcal{F}_{\mathsf{GEQ}}$ as being the shares of the corrupted party for x and y (note that the result of $\mathcal{F}_{\mathsf{GEQ}}$ only depends on the value of $x - y \mod q$). $\mathcal{S}$ is also able to fix the output share of the corrupted party in $\mathcal{F}_{\mathsf{GEQ}}$ so that it matches the one in the instance of π_{GEQ}. This is a perfect simulation strategy, no environment $\mathcal{Z}$ can distinguish the ideal and real worlds and therefore π_{GEQ} UC-realizes $\mathcal{F}_{\mathsf{GEQ}}$.

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

$\mathcal{F}_{\mathsf{GEQ}}$ runs with Alice and Bob and is parametrized by the bit length λ of the ring $\mathbb{Z}_q$ (i.e., $q = 2^\lambda$). It receives as input the secret shared values x and y, which are guaranteed to be such that $|x - y| < 2^{\lambda-1}$ (as integers).

Input: Upon receiving a message from Alice or Bob with its share of $[\![x]\!]_q$ and $[\![y]\!]_q$, 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 the values x and y, and compute diff $= x - y$. If diff represents a negative number, distribute a new secret sharing $[\![0]\!]_2$; otherwise a new secret sharing $[\![1]\!]_2$. 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.

From the UC-security of the building blocks, and noting that $\mathcal{S}$ being responsible for simulating $\mathcal{F}_{\mathsf{TI}}^{\mathcal{D}}$ in the internal simulations of the protocol π_{PPNBC} for the adversary $\mathcal{A}$ is able to extract all inputs of the corrupted party and forward them to the functionality $\mathcal{F}_{\mathsf{PPNBC}}$, it follows straightforwardly that π_{PPNBC} UC-realizes functionality $\mathcal{F}_{\mathsf{PPNBC}}$.

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

Input: Upon receiving a message from Alice with her inputs $(a_1, \ldots, a_m)$ or from Bob with his inputs $(b_1, \ldots, b_n)$ and $(\log(P(c_j)), \log(P(b_1|c_j)), \ldots, \log(P(b_n|c_j)),$ $\log(1 - P(b_1|c_j)), \ldots, \log(1 - P(b_n|c_j)))$ for each class c_j, record the values, 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_{PPNBC} using the secret sharings. Let $[\![c]\!]_2$ be the result. 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 $[\![c]\!]_2$. The output shares are delivered to the parties.

V. EXPERIMENTAL RESULTS

To evaluate the proposed protocol in a use case for spam detection, we use the SMS Spam Collection Data Set from the UC Irvine Machine Learning Repository.[2] This database contains a set of tagged SMS messages that have been collected for SMS spam research. It contains a set of 5574 SMS messages in English tagged as legitimate (ham) or spam. The files contain one message per line, where each line is composed of two columns: the first contains the label (ham or spam) and the second contains the raw text (see examples in Figure 2). The data set has 747 spam SMS messages and 4827 ham SMS messages, that is, 13.4% of the SMSes are spams and 86.6% are hams.

Table I shows the distribution of tokens/unigrams in the data set. As we can see, the data set has a total of 81175 tokens. When training a spam classifier, techniques can be used to reduce this set of tokens in order to improve the performance of the protocol in terms of accuracy, runtime or other metrics.

TABLE I
TOKEN STATISTICS [5].

Tokens in Hams	63632
Tokens in Spams	17543
Total of Tokens	81175
Average Tokens per Ham	13.18
Average Tokens per Spam	23.48
Average Tokens per Message	14.56

[2]https://archive.ics.uci.edu/ml/datasets/sms+spam+collection

A. Training phase

In the classification phase, Bob already has the ML model. The model was generated using the following steps:

1) Bob takes the SMS Spam Collection Data Set and parses each line into unigrams. The letters are converted to lower case and everything other than letters is deleted.
2) To have higher accuracy and improve the runtime of the algorithm, we used the stemming and stop words techniques. Stemming is the process of reducing the inflection of words in their roots, such as mapping a group of words to the same stem even if the stem itself is not a valid word in the language. For example, likes, liked, likely and liking reduce to the stem like; retrieval, retrieved, retrieves reduce to the stem retrieve; trouble, troubled and troubles reduce to the stem troubl. Stop words concerns filtering out words that can be considered irrelevant to the classification task such as: the, a, an, in, to, for.
3) The remaining unigrams are inserted in a Bag of Words (BoW). A BoW is created for the ham category and another for the spam category. Each BoW contains the unigrams and their corresponding frequency counters.
4) Based on the frequency counters, we remove the less frequent words in order to decrease the runtime of our privacy-preserving solution. We will address this parameter later when we detail the trade-off between accuracy and efficiency.
5) Bob computes the logarithm of the class prior probability for each class c:

$$\log(P(c)) = \log\left(\frac{|\text{training examples} \in c|}{|\text{examples in the training set}|}\right). \quad (1)$$

6) Bob computes the logarithm of the probability of each word by class. To compute the probability we have to find the average of each word for a given class. For the class c and the word i, the average is given by:

$$\log(P(i|c)) = \log\left(\frac{|\text{word "i" in class } c|}{|\text{words in class } c|}\right). \quad (2)$$

However, as some words can have 0 occurrences, we use Laplace Smoothing:

$$\log(P(i|c)) = \log\left(\frac{|\text{word "i" in class } c| + 1}{|\text{words in class } c| + |V|}\right), \quad (3)$$

where $|V|$ is the size of the vocabulary, i.e., all unique words in the training data set regardless of the class.

In Equations 1 and 3, before computing the logarithm we need to scale the result of the division to convert it to integers. Before inputing this model into our privacy-preserving protocol, we need to convert any fixed precision real number into an integer. In order to do so, we follow section II-B, we pick up a value of a equal to 34. With the values of $\log(P(c))$ and $\log(P(i|c))$ computed, the model is generated. Note that only Bob is involved in training the model.

Table II shows the distribution of tokens/unigrams in the data set after performing the training phase. Compared to the

ham	Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...
ham	Ok lar... Joking wif u oni...
spam	Congrats! 1 year special cinema pass for 2 is yours. call 09061209465 now! C Suprman V, Matrix3, StarWars3, etc all 4 FREE! bx420-ip4-5we. 150pm. Dont miss out!
ham	Yup next stop
spam	Please call our customer service representative on 0800 169 6031 between 10am-9pm as you have WON a guaranteed £1000 cash or £5000 prize!
ham	Waiting for your call

Fig. 2. Some examples of tagged SMS messages from the SMS Spam Collection Data Set.

Table I, there was a reduction of over 30 thousand tokens. Besides, we can see that we have less than 6000 unique tokens.

TABLE II
TOKEN STATISTICS AFTER THE TRAINING PHASE.

Tokens in Hams	38469
Tokens in Spams	10981
Total of Tokens	49450
Average Tokens per Ham	7.97
Average Tokens per Spam	14.7
Average Tokens per Message	8.87
Unique Tokens in Hams	5950
Unique Tokens in Spams	1883
Unique Tokens in the Data Set	5950

B. Cryptographic Engineering

Our solution to secure Naive Bayes classification is implemented in Rust using an up-to-date version of the RustLynx framework (available at https://bitbucket.org/uwtppml/rustlynx/src/master/), which was used in [25] to achieve a state-of-the-art implementation of secure logistic regression training. The supported primitives are, to the best of our knowledge, the fastest available for two-party computation in the honest-but-curious setting when performed in a local area network.

a) Parallel Data Transfer: Instead of atomic sending and receiving queues as might be utilized in a general-purpose multi-threaded network application, we associate each thread (for a fixed threadpool size) with a port in a given port range such that the i-th thread executing in the P_0 process will only exchange data with the i-th thread of the P_1 process. We base this choice on the observation that MPC operations are symmetric, so there is never an instance where the receiver does not know the length, intent, or timing of a message from the sender – situations for which a more complex, and slower, messaging system would be necessary. An additional benefit of this structure is that the packets require no header to denote the sender, thread ID, or length of the body. Based on empirical testing on $\mathbb{Z}_{2^{64}}$ multiplication with an optimized threadpool size, this method yields a $6\times$ improvement over an architecture with atomic queues.

b) Operations in $\mathbb{Z}_{2^\lambda}$: RustLynx supports arithmetic over $\mathbb{Z}_{2^{64}}$. This particular bit length is chosen because (1) it is sufficiently large to represent realistic data in a fixed-point form and (2) it aligns with a primitive data type, meaning that modular arithmetic operations can be performed implicitly by permitting integer overflow.

c) Operations in $\mathbb{Z}_2$: We represent $\mathbb{Z}_2$ shares in groupings of 128 as the individual bits of Rust's unsigned 128-bit integer primitive. Doing so allows for local operations on the entire group of secrets to share Arithmetic Logic Unit (ALU) cycles and to be loaded, copied, and iterated quickly. The downside of this design choice is that sending m $\mathbb{Z}_2$ shares corresponds to $16 \cdot \lceil m/128 \rceil$ bytes of data transfer, which, in the worst case, is 15 bytes larger than the most compact possible representation of m bits (that is, using groups of 8). Based on empirical testing, the performance loss for MPC primitives is affected significantly more by wasting time on local operations than wasting a small amount of bandwidth. So, the largest available primitive data type was chosen to group $\mathbb{Z}_2$ shares.

C. Evaluation

We ran our experiments on Amazon Web Services (AWS) using two c5.9x-large EC2 instances with 36 vCPUs, 72.0 GiB of memory and 32 threads. Each of the parties ran on separate machines (connected over a Gigabit Ethernet LAN), which means that the results in Table IV cover the communication time in addition to the computation time. All experiments were repeated one-hundred times and averaged to minimize the variance caused by large thread counts.

We evaluate PPNBC using 5-fold cross validation over the entire corpus of 5574 SMS. For each unigram in Alice's set $A = \{a_1, a_2, \cdots, a_m\}$ and each unigram in Bob's set $B = \{b_1, b_2, \cdots, b_n\}$, we apply the hash function SHA-256 (and truncate the result) to transform each one into a bit-string of size $\ell = 14$.

We evaluated our solution for $m = \{8, 16, 50, 160\}$ and $n = \{369, 484, 688, 5200\}$. Note that the n value affects the accuracy and running time. The n values were defined based on the frequency of tokens appearing in the training data set: 688 tokens appeared more than 9 times; 484 tokens appeared more than 14 times; and 369 tokens appeared more

than 19 times. We noticed a significant degradation of the False Positive Rate (FPR) when further reducing n. The values of m were defined based on the size of our messages. Note that m determines the number of tokens in the message and not the number of characters. Also, we should mention that some messages in our data set consisted of multiple SMSes concatenated. Our maximum value of n (160 tokens) is twice the maximum message found in our data set. We recall that a single SMS has a 160 7-bit characters maximum. The average lengths found for SMS classified as ham or spam in our data set are shown in Table II.

We evaluate the proposed protocol in a use case for SMS spam detection, however our PPNBC can be used in any other scenario in which the Naive Bayes classifier can be employed. It is important to 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 a privacy-preserving Naive Bayes classifier based on MPC is feasible in practice. Despite this, as shown in Table III, the protocol achieves good results when compared to the best result presented by Almeida *et al.* [5], where the data set is proposed. They reach an accuracy equal to 97.64%, a false positive rate (FPR) equal to 16.9% and a false negative rate (FNR) equal to 0.18% using a SVM classifier. In our best scenario ($n = 5200$), we have an accuracy equal to 96.8%, FPR equal to 17.94% and FNR equal to 0.87%. We remark that there is little variation in accuracy and FNR when using smaller values of n.

TABLE III
ACCURACY RESULTS USING 5-FOLD CROSS-VALIDATION OVER THE
CORPUS OF 5574 SMS. FPR IS THE FALSE POSITIVE RATE AND FNR IS
THE FALSE NEGATIVE RATE.

Dictionary size	FNR	FPR	Accuracy
n=369	0.79%	28.52%	95.5%
n=484	0.89%	22.22%	96.2%
n=688	0.87%	21.15%	96.4%
n=5200	0.87%	17.94%	96.8%

Table IV reports the runtime of our PPNBC for different sizes of m and n, where n is the size of the dictionary, that is, the amount of unigrams belonging to Bob's trained model and m is the number of unigrams present in Alice's SMS. The feature vector extraction (Extr) runtime considers the time required to execute the Protocols π_{FE} and π_{2toQ} in the steps 1 and 2 of Protocol π_{PPNBC}. The runtime for classification (Class) considers the remaining steps of the Protocol π_{PPNBC}. And, the total runtime is Extr + Class. We can see that the runtime for classification is independent of the size of m, and is based only on the size n, and even for $n = 5200$ features/unigrams it only takes $48\,$ms. The feature vector extraction (Extr) runtime depends on both m and n. For $n = 5200$ and $m = 160$, the feature extraction takes less than $290\,$ms, while for $n = 369$ and $m = 8$ it just takes $11\,$ms. The total runtime takes a maximum of $334\,$ms to classify a SMS with 160 unigrams using a dictionary with 5200 unigrams, and just $21\,$ms to classify a SMS with 8 unigrams using a dictionary with 369 unigrams.

As we can notice from Table IV, the feature extraction is the part that spends the most time because $m \times n$ secure equality tests of bit strings are required, which are based on secure multiplications. As discussed by Reich *et al.* [52], the number of secure equality tests needed could possibly be reduced if Alice and Bob first map using the same hash function each of their bitstrings to t buckets, $A_1, A_2, \cdots, A_t$ for Alice and $B_1, B_2, \cdots, B_t$ for Bob. Then, only the bitstrings belonging to A_i would need to be compared with the bitstrings belonging to B_i.

To use buckets, each a_m and b_n element is hashed, and the result is divided into two parts, where the first q bits indicate which bucket the element belongs to and the other r bits are stored, thus $t = 2^q$. To hide how many elements are mapped to each bucket, as this can leak information about the distribution of the elements, the empty spots of each bucket must be filled up with dummy elements. Thus, considering s_1 the size of each bucket A_t and s_2 the size of each bucket B_t, the extraction feature protocol will need $t \times s_1 \times s_2$ equality tests, which can be substantially smaller than $n \times m$ needed previously. It is important to note that these dummy elements do not modify the accuracy (or any other metrics) of the classification, because when generating Bob's model, for each dummy element, the probability of the element to occur for each class is defined as 0, that is, it does not impact $\sum_{k=1}^{d} \log(P(x_k|c))$. In our case, since the values of m and n are not large, there is no significant difference between using buckets or not. Therefore, we use the original version (without buckets), as in this case there is no probability of information being leaked due to buckets' overflow.

Finally, we remark that, for the sake of evaluating our solution, we have selected values of n (the dictionary size) that directly depend on the frequency of tokens. That is not necessary in general.

VI. RELATED WORK

Privacy-preserving versions of ML classifiers were first addressed by Lindell and Pinkas [45]. They used MPC to build a secure ID3 decision tree where the training set is distributed between two parties. Most of the literature on privacy-preserving ML focus on the training phase, and include secure training of ML algorithms such as Naive Bayes [62], [58], [51], decision tree [45], [11], [26], logistic regression [16], [48], [25], linear regression [24], [48], [2], neural networks [56], [48], [61] and SVM [59]. Regarding privacy-preserving classification/inference/prediction, most works focused on secure neural network inference, e.g., [7], [36], [48], [46], [54], [40], [61], [3], [53], [19], [47], [44], [50]. Far less works focus on privacy in the classification/prediction phase of other algorithms. De Hoogh et al. [26] has a protocol for privacy-preserving scoring of decision trees. Bost *et al.* [10] proposed privacy-preserving classification protocols for hyperplane-based classifiers, Naive Bayes and decision trees where the description of the features (the dictionary in our implementation) was supposed to be public. David et al. [21] presented protocols for privacy-preserving classification with hyperplane-based classifiers and Naive Bayes, again, the classifier features were supposed to be publicly known. Our solution guarantees the privacy of the

TABLE IV
TOTAL RUNTIME IN MILLISECONDS (TOTAL) NEEDED TO SECURELY CLASSIFY A SMS WITH OUR PROPOSAL. WE DIVIDED IT IN THE TIME NEEDED FOR FEATURE VECTOR EXTRACTION (EXTR) AND THE TIME FOR CLASSIFICATION (CLASS). n IS THE SIZE OF THE DICTIONARY, THAT IS, THE AMOUNT OF UNIGRAMS BELONGING TO BOB'S TRAINED MODEL AND m IS THE AMOUNT OF UNIGRAMS PRESENT IN ALICE'S SMS.

Dictionary size	SMS size											
	m=8			m=16			m=50			m=160		
	Extr	Class	Total	Extr	Class	Total	Extr	Class	Total	Extr	Class	Total
n=369	11 ms	10 ms	21 ms	25 ms	10 ms	35 ms	102 ms	10 ms	112 ms	111 ms	10 ms	121 ms
n=484	12 ms	10 ms	22 ms	26 ms	10 ms	36 ms	103 ms	10 ms	113 ms	124 ms	10 ms	134 ms
n=688	20 ms	11 ms	33 ms	36 ms	11 ms	47 ms	106 ms	11 ms	117 ms	136 ms	11 ms	147 ms
n=5200	77 ms	48 ms	125 ms	89 ms	48 ms	137 ms	140 ms	48 ms	188 ms	286 ms	48 ms	334 ms

dictionary. Khedr *et al.* [42] proposed a secure NB classifier based on Fully Homomorphic Encryption (FHE). De Cock et al. [23] presented private scoring protocols for decision trees and hyperplane-based classifiers. Fritchman et al. [35] presented a solution for private scoring of tree ensembles.

Regarding privacy-preserving text classification, Costantino et al. [17] presented a proposal based on homomorphic encryption that takes 19 minutes to classify a tweet, with 19 features and 76 minutes to classify an email with 17 features. In addition to the high runtime, Bob learns which of his lexicon's words are present in Alice's tweets. To the best of our knowledge, the recent work by Reich et al. [52] was the first work to present solutions for privacy-preserving feature extraction and classification of unstructured texts. The best time they got, 4.5 seconds, was using unigrams, 50 features and logistic regression with an accuracy of 72.4%. The highest accuracy they obtained was 74.4% using unigram and bigrams, 500 features and AdaBoost with a running time equal to 28.3s.

As the protocols presented by Reich et al. [52], our solution does not leak any information about Alice's words to Bob neither the words of Bob's model for Alice, and classifies an SMS as ham or spam (even for a model with 5200 features) in less than 0.3s, in the worst case, and less than 0.022s for an average message of our data set, while using the same type of machines that they used. Our results include communication and computation times.

More recently, Badawi *et. al* proposed a protocol for privacy-preserving text classification based on fully homomorphic encryption [4]. They obtained a highly efficient, GPU-accelerated implementation that improves the state-of-the-art of FHE based inference by orders of magnitude. A GPU equipped machine can compute the private classification of a text message in about 0.17 seconds in their implementation. This time does not include the communication time to send the encrypted text from the client to the server and the time to receive the result. In a Gigabit Ethernet network, that would probably add anything between 0.3s to 0.5s to their total running time because of the ciphertext expansion resulting from the use of FHE.

VII. CONCLUSION

Privacy-preserving machine learning protocols are powerful solutions to perform operations on data while maintaining the privacy of the data. To the best of our knowledge, we propose the first privacy-preserving Naive Bayes classifier with private feature extraction. No information is revealed regarding either

Bob's model (including which words belong to the model) or the words contained in Alice's SMS. Our Rust implementation provides a fast and secure solution for the classification of unstructured text. Applying our solution to the case of spam detection, we can classify an SMS as spam or ham in less than 340 ms in the case where the dictionary size of Bob's model includes all words ($n = 5200$) and Alice's SMS has at most $m = 160$ unigrams. In the case with $n = 369$ and $m = 8$ (the average of a spam SMS in the database), our solution takes only 21 ms. Besides, the accuracy is practically the same as performing the Naive Bayes classification in the clear. It is important to note that our solution can be used in any application where Naive Bayes can be used. Thus, we believe that our solution is practical for the privacy-preserving classification of unstructured text. To the best of our knowledge, our solution is the fastest SMC based solution for private text classification.

REFERENCES

[1] Samuel Adams, Chaitali Choudhary, Martine De Cock, Rafael Dowsley, David Melanson, Anderson Nascimento, Davis Railsback, and Jianwei Shen. Secure Training of Decision Tree based Models over Continuous Data. Manuscript, 2020.

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

[3] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. QUOTIENT: Two-party secure neural network training and prediction. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 1231–1247. ACM Press, November 11–15, 2019.

[4] Ahmad Al Badawi, Louie Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. Privft: Private and fast text classification with homomorphic encryption. *IEEE Access*, 8:226544–226556, 2020.

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

[6] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *45th Annual Symposium on Foundations of Computer Science*, pages 186–195, Rome, Italy, October 17–19, 2004. IEEE Computer Society Press.

[7] Mauro Barni, Pierluigi Failla, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Privacy-Preserving ECG Classification With Branching Programs and Neural Networks. *IEEE Trans. Information Forensics and Security*, 6(2):452–468, 2011.

[8] 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.

[9] Donald Beaver. Commodity-Based Cryptography (Extended Abstract). In *STOC*, pages 446–455. ACM, 1997.

[10] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. In *NDSS*. The Internet Society, 2015.

[11] Justin Brickell and Vitaly Shmatikov. Privacy-Preserving Classifier Learning. In *Financial Cryptography*, volume 5628 of *Lecture Notes in Computer Science*, pages 128–147. Springer, 2009.

[12] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[13] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology – CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 19–40, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Heidelberg, Germany.

[14] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing*, pages 494–503, Montréal, Québec, Canada, May 19–21, 2002. ACM Press.

[15] Raffaele Cappelli, Matteo Ferrara, and Davide Maltoni. Minutia Cylinder-Code: A New Representation and Matching Technique for Fingerprint Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(12):2128–2141, 2010.

[16] Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *NIPS*, pages 289–296. Curran Associates, Inc., 2008.

[17] Gianpiero Costantino, Antonio La Marra, Fabio Martinelli, Andrea Saracino, and Mina Sheikhalishahi. Privacy-preserving text mining as a service. In *ISCC*, pages 890–897. IEEE Computer Society, 2017.

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

[19] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. Secure evaluation of quantized neural networks. *Proceedings on Privacy Enhancing Technologies*, 2020(4):355–375, October 2020.

[20] Bernardo David and Rafael Dowsley. Efficient composable oblivious transfer from CDH in the global random oracle model. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20: 19th International Conference on Cryptology and Network Security*, volume 12579 of *Lecture Notes in Computer Science*, pages 462–481, Vienna, Austria, December 14–16, 2020. Springer, Heidelberg, Germany.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] Martine De Cock, Rafael Dowsley, Anderson C. A. Nascimento, Davis Railsback, Jianwei Shen, and Ariel Todoki. High Performance Logistic Regression for Privacy-Preserving Genome Analysis. *To appear at BMC Medical Genomics. Available at https://arxiv.org/abs/2002.05377*, 2021.

[26] Sebastiaan de Hoogh, Berry Schoenmakers, Ping Chen, and Harm op den Akker. Practical secure decision tree learning in a teletreatment application. In Nicolas Christin and Reihaneh Safavi-Naini, editors, *FC 2014: 18th International Conference on Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 179–194, Christ Church, Barbados, March 3–7, 2014. Springer, Heidelberg, Germany.

[27] Jia Deng, Alexander C. Berg, Kai Li, and Fei-Fei Li. What Does Classifying More Than 10,000 Image Categories Tell Us? In *ECCV (5)*, volume 6315 of *Lecture Notes in Computer Science*, pages 71–84. Springer, 2010.

[28] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In Yuval Ishai, editor, *TCC 2011: 8th Theory of Cryptography Conference*, volume 6597 of *Lecture Notes in Computer Science*, pages 164–181, Providence, RI, USA, March 28–30, 2011. Springer, Heidelberg, Germany.

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

[30] 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.

[31] Rafael Dowsley, Jörn Müller-Quade, and Tobias Nilges. Weakening the isolation assumption of tamper-proof hardware tokens. In Anja Lehmann and Stefan Wolf, editors, *ICITS 15: 8th International Conference on Information Theoretic Security*, volume 9063 of *Lecture Notes in Computer Science*, pages 197–213, Lugano, Switzerland, May 2–5, 2015. Springer, Heidelberg, Germany.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin E. Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *ICML*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 201–210. JMLR.org, 2016.

[37] Dennis Hofheinz and Jörn Müller-Quade. Universally composable commitments using random oracles. In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 58–76, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.

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

[39] 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.

[40] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security Symposium*, pages 1651–1669. USENIX Association, 2018.

[41] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In Moni Naor, editor, *Advances in Cryptology – EUROCRYPT 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 115–128, Barcelona, Spain, May 20–24, 2007. Springer, Heidelberg, Germany.

[42] Alhassan Khedr, P. Glenn Gulak, and Vinod Vaikuntanathan. SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers. *IEEE Trans. Computers*, 65(9):2848–2858, 2016.

[43] Neeraj Kumar, Alexander C. Berg, Peter N. Belhumeur, and Shree K. Nayar. Describable Visual Attributes for Face Verification and Image Search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(10):1962–1977, 2011.

[44] Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow: Secure TensorFlow inference. In *2020 IEEE Symposium on Security and Privacy*, pages 336–353, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

[45] Yehuda Lindell and Benny Pinkas. Privacy Preserving Data Mining. In *CRYPTO*, volume 1880 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2000.

[46] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *ACM Conference on Computer and Communications Security*, pages 619–631. ACM, 2017.

[47] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In Srdjan Capkun and Franziska Roesner, editors,

USENIX Security 2020: 29th USENIX Security Symposium, pages 2505–2522. USENIX Association, August 12–14, 2020.

[48] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *IEEE Symposium on Security and Privacy*, pages 19–38. IEEE Computer Society, 2017.

[49] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571, Santa Barbara, CA, USA, August 17–21, 2008. Springer, Heidelberg, Germany.

[50] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. CrypTFlow2: Practical 2-party secure inference. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 20: 27th Conference on Computer and Communications Security*, pages 325–342, Virtual Event, USA, November 9–13, 2020. ACM Press.

[51] Olivier Regnier-Coudert and John A. W. McCall. Privacy-preserving approach to bayesian network structure learning from distributed data. In *GECCO (Companion)*, pages 815–816. ACM, 2011.

[52] Devin Reich, Ariel Todoki, Rafael Dowsley, Martine De Cock, and Anderson C. A. Nascimento. Privacy-Preserving Classification of Personal Text Messages with Secure Multi-Party Computation. In *NeurIPS*, pages 3752–3764, 2019.

[53] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin E. Lauter, and Farinaz Koushanfar. XONN: XNOR-based oblivious deep neural network inference. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019: 28th USENIX Security Symposium*, pages 1501–1518, Santa Clara, CA, USA, August 14–16, 2019. USENIX Association.

[54] M. Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M. Songhori, Thomas Schneider, and Farinaz Koushanfar. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *AsiaCCS*, pages 707–721. ACM, 2018.

[55] 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.

[56] Reza Shokri and Vitaly Shmatikov. Privacy-Preserving Deep Learning. In *ACM Conference on Computer and Communications Security*, pages 1310–1321. ACM, 2015.

[57] 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.

[58] Jaideep Vaidya, Murat Kantarcioglu, and Chris Clifton. Privacy-preserving Naïve Bayes classification. *VLDB J.*, 17(4):879–898, 2008.

[59] Jaideep Vaidya, Hwanjo Yu, and Xiaoqian Jiang. Privacy-preserving SVM classification. *Knowl. Inf. Syst.*, 14(2):161–178, 2008.

[60] Mike Voets, Kajsa Møllersen, and Lars Ailo Bongo. Replication study: Development and validation of deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. *CoRR*, abs/1803.04337, 2018.

[61] Sameer Wagh, Divya Gupta, and Nishanth Chandran. SecureNN: 3-party secure computation for neural network training. *Proceedings on Privacy Enhancing Technologies*, 2019(3):26–49, July 2019.

[62] Rebecca N. Wright and Zhiqiang Yang. Privacy-preserving Bayesian network structure computation on distributed heterogeneous data. In *KDD*, pages 713–718. ACM, 2004.