ppAURORA: Privacy Preserving Area Under Receiver Operating Characteristic and Precision-Recall Curves with Secure 3-Party Computation

Ali Burak Ünal  
Methods in Medical Informatics  
Department of Computer Science  
University of Tuebingen  
Germany  
uenal@informatik.uni-tuebingen.de

Nico Pfeifer *  
Methods in Medical Informatics  
Department of Computer Science  
University of Tuebingen  
Germany  
pfeifer@informatik.uni-tuebingen.de

Mete Akgün †  
Methods in Medical Informatics  
Department of Computer Science  
University of Tuebingen  
Germany  
akguen@informatik.uni-tuebingen.de

Abstract

Computing an AUC as a performance measure to compare the quality of different machine learning models is one of the final steps of many research projects. Many of these methods are trained on privacy-sensitive data and there are several different approaches like \( \epsilon \)-differential privacy, federated machine learning and methods based on cryptographic approaches if the datasets cannot be shared or evaluated jointly at one place. In this setting, it can also be a problem to compute the global performance measure like an AUC, since the labels might also contain privacy-sensitive information. There have been approaches based on \( \epsilon \)-differential privacy to deal with this problem, but to the best of our knowledge, no exact privacy preserving solution has been introduced. In this paper, we propose an MPC-based framework, called ppAURORA, with private merging of sorted lists and novel methods for comparing two secret-shared values, selecting between two secret-shared values, converting the modulus, and performing division to compute the exact AUC as one could obtain on the pooled original test samples. With ppAURORA computation of the exact area under precision-recall curve and receiver operating characteristic curve is even possible when ties between prediction confidence values exist. To show the applicability of ppAURORA, we use it to evaluate a model trained to predict acute myeloid leukemia therapy response and we also assess its scalability via experiments on synthetic data. The experiments show that we efficiently compute exactly the same AUC with both evaluation metrics in a privacy preserving manner as one can obtain on the pooled test samples in the plaintext domain. Our solution provides security against semi-honest corruption of at most one of the servers performing the secure computation.

*Also associated with Statistical Learning in Computational Biology, Max Planck Institute for Informatics, Saarbrücken, Germany  
†Also associated with Translational Bioinformatics, University Hospital Tuebingen, Tuebingen, Germany

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.
1 Introduction

Recently, privacy preserving machine learning studies aimed at protecting sensitive information during training and/or testing of a model in scenarios where data is distributed between different sources and cannot be shared in plaintext \cite{Mohassel and Zhang, 2017, Wagh et al., 2018, Juvekar et al., 2018, Mohassel and Rindal, 2018, Ünal et al., 2019, Byali et al., 2020, Patra and Suresh, 2020}. However, privacy protection in the computation of the area under curve (AUC), which is one of the most preferred methods to compare different machine learning models with binary outcome, has not been addressed sufficiently. Even though there are no studies in the literature enabling such a computation for the precision-recall (PR) curve, there are several differential privacy based approaches in the literature for the receiver operating characteristic (ROC) curve \cite{Chaudhuri and Vinterbo, 2013, Boyd et al., 2013, Chen et al., 2015}. Briefly, they protect the privacy of the data by introducing noise into the computation so that one cannot obtain the original data employed in the computation. However, due to the nature of differential privacy, the resulting AUC is different from the one which could be obtained by using non-perturbed prediction confidence values (PCVs). For private computation of the exact AUC, there exists no approach in the literature to the best of our knowledge.

In this paper, we propose the privacy preserving area under receiver operating characteristic and precision-recall curves (ppAURORA) based on a secure 3-party computation framework to address the necessity of an efficient, private and secure computation of the exact AUC. We compute the area under the PR curve (AUPR) and ROC curve (AUROC) with ppAURORA. We address two different cases of ROC curve in ppAURORA by two different versions of AUROC computation. The first one is designed for the computation of the exact AUC by using PCVs with no tie. In case of a tie of PCVs of samples from different classes, it just approximates the metric based on the order of the samples, having a problem when values of both axes change at the same time. In order to compute the exact AUC even in case of a tie, we introduce the second version with a slightly higher communication cost than the first approach. Along with the AUC, both are capable of protecting the information of the number of samples belonging to the classes from all participants of the computation, which could be used to obtain the order of the labels of the PCVs \cite{Whitehill, 2019}. Furthermore, since we do not provide the data sources with the ROC curve, they cannot regenerate the underlying true data. Therefore, both versions are secure against such attacks \cite{Matthews and Harel, 2013}. We utilized the with-tie version of AUROC computation to compute the AUPR since the values of both axes can change at the same time even if there is no tie.

We introduce a novel 3-party computation framework to achieve privacy preserving AUC computation with ppAURORA. The framework consists of privacy preserving sorting and four operations with novel and efficient versions, which are select share (MUX), modulus conversion (MC), compare (CMP) and division (DIV). MUX is designed to select one of two secret shares based on a secret shared bit value. MC privately converts the ring of a secret shared value from $2^{ℓ} - 1$ to $2^{ℓ}$. CMP compares two secret shared values, determines if the first argument is larger than the second one without revealing values and splits the result in a secret shared way. DIV performs the division of two secret shared values without sacrificing the privacy of the values. Note that our new DIV is specifically customized for efficient and secure AUC computation.

2 Scenarios

In this section, we describe the scenarios at which ppAURORA is applicable. Note that it is not limited to these scenarios.

**End-to-end MPC-based Collaborative Learning:** Recently, researchers proposed multi-party computation (MPC) based training and testing of several machine learning algorithms \cite{Mohassel and Zhang, 2017, Wagh et al., 2018, Damgård et al., 2019}. Their approaches can train the model privately and collaboratively, and make predictions on the test data of the data sources involved in the computation. However, the privacy preserving collaborative evaluation of the model is lacking. To fulfill such a gap, one can integrate ppAURORA at the end of the process once the PCVs are secret shared among two computing parties. One possible scenario would be that one wants to build a model for predicting hospitalization times for COVID-19 patients. Usually, the personal data cannot be shared easily, due to the private nature of the data, but even sharing the hospitalization times might be problematic in case the hospitals do not want to allow competitors to learn about this piece of information. Nevertheless, one can assume that a model built on data from many hospitals will perform much better than models built on individual datasets and that the global AUC will allow for a better model selection than an average of locally computed AUCs. This privacy preserving global AUC computation can be performed with our ppAURORA.

**Evaluation of Models Trained by Federated Learning:** In some cases, the data is not allowed to be shared at all. For such cases, federated learning has been widely utilized to train a collaborative machine learning model without gathering the data in one place. Each data source updates the model by its own data in an online learning process and passes the model to the next data source until the model converges or the iteration limit is reached. Once the data sources obtain the trained model, they make the predictions of their own test data. In order to collaboratively evaluate the performance of the model, they can utilize ppAURORA.
3 Preliminaries

Security Model: In this study, we prove the full security of our solution (i.e., privacy and correctness) in the presence of semi-honest adversaries that follow the protocol specification, but try to learn information from the execution of the protocol. We consider a scenario where a semi-honest adversary corrupts a single server and an arbitrary number of data owners in the simulation paradigm [Lindell, 2017] and [Canetti, 2001] where two worlds are defined: the real world where parties run the protocol without any trusted party, and the ideal world where parties make the computation through a trusted party. Security is modeled as the view of an adversary called a simulator \( S \) in the ideal world, who cannot be distinguished from the view of an adversary \( A \) in the real world. The universal composability framework [Canetti, 2001] introduces an adversarial entity called environment \( Z \), which gives inputs to all parties and reads outputs from them. The environment is used in modeling the security of end-to-end protocols where several secure protocols are used arbitrarily. Security here is modeled as no environment can distinguish if it interacts with the real world and the adversary \( A \) or the ideal world and the simulator \( S \). We also provide privacy in the presence of a malicious adversary corrupting any single server, which is formalized by [Araki et al., 2016]. The privacy is formalized by saying that a malicious party, which arbitrarily deviates from the protocol description, cannot learn anything about the inputs and outputs of the honest parties.

Notations: In our secure protocols, we use additive secret sharing over three different rings \( \mathbb{Z}_L, \mathbb{Z}_K \) and \( \mathbb{Z}_P \) where \( L = 2^k, K = 2^{k-1}, P = 67 \) and \( \ell = 64 \). We denote two shares of \( x \) over \( \mathbb{Z}_L, \mathbb{Z}_K \) and \( \mathbb{Z}_P \) with \((\langle x \rangle_0, \langle x \rangle_1), \langle x \rangle^B_i, \langle x \rangle^F_i \) and \((\langle x \rangle^B_0, \langle x \rangle^B_1)\), respectively. If a value \( x \) is shared over the ring \( \mathbb{Z}_P \), each bit of \( x \) is additively shared in \( \mathbb{Z}_P \). This means \( x \) is shared as a vector of 64 shares where each share takes a value between 0 and 66. We also use boolean sharing of a single bit which is denoted with \((\langle x \rangle^B_0, \langle x \rangle^B_1)\).

3.1 Secure Multi-party Computation

Secure multi-party computation was proposed in the 1980s [Yao, 1986, Goldreich et al., 1987]. These studies showed that multiple parties can compute any function on inputs without learning anything about the inputs of the other parties. Let us assume that there are \( n \) parties \( I_1, \ldots, I_n \) and \( I_i \) has a private input \( x_i \) for \( i \in \{1, \ldots, n\} \). All parties want to compute the arbitrary function \((y_1, \ldots, y_n) = f(x_1, \ldots, x_n)\) and get the result \( y_i \). MPC allows the parties to compute the function through an interactive protocol and \( I_i \) to learn only \( y_i \).

We first explain the 2-out-of-2 additive secret sharing and how addition (ADD) and multiplication (MUL) are computed. In additive secret sharing, an \( \ell \)-bit value \( x \) is shared additively in the ring \( \mathbb{Z}_L \) as the sum of two values. For \( \ell \)-bit secret sharing of \( x \), we have \( \langle x \rangle_0 + \langle x \rangle_1 \equiv x \mod L \) where \( I_i \) knows only \( \langle x \rangle_i \) and \( i \in \{0, 1\} \). All arithmetic operations are performed in the ring \( \mathbb{Z}_L \). For additive secret sharing, we use protocols based on Beaver’s multiplication triples [Beaver, 1991].

Addition: \( \langle z \rangle = \langle x \rangle + \langle y \rangle \). \( I_i \) locally computes \( \langle z \rangle_i = \langle x \rangle_i + \langle y \rangle_i \). In order to compute the addition of a shared value \( \langle x \rangle \) and a constant \( c \), \( I_i \) locally computes \( \langle z \rangle_i = \langle x \rangle_i + c \) and \( I_{1-i} \) locally computes \( \langle z \rangle_{1-i} = \langle x \rangle_{1-i} \).

Multiplication: \( \langle z \rangle = \langle x \rangle \cdot \langle y \rangle \). Multiplication is performed using a pre-computed multiplication triple \( \langle c \rangle_i = \langle a \rangle_i \cdot \langle b \rangle_i \) [Beaver, 1991]. \( I_i \) computes \( \langle c \rangle_i = \langle x \rangle_i - \langle a \rangle_i \) and \( \langle f \rangle_i = \langle y \rangle_i - \langle b \rangle_i \). \( I_i \) sends \( \langle c \rangle_i \) and \( \langle f \rangle_i \) to \( I_{1-i} \). \( I_i \) reconstructs \( e \) and \( f \), and then computes \( \langle z \rangle_i = i \cdot e + f \cdot (a)_i + e \cdot (b)_i + (c)_i \). The computation of the multiplication triple is performed via homomorphic encryption or oblivious transfer. \( I_i \) cannot perform multiplication locally.

3.2 Area Under Curve

One of the most common ways to summarize the plot-based model evaluation metrics is area under curve (AUC) which measures the area under the curve. It is applicable to various different evaluation metrics. In this study, we employ AUC to measure the area under the ROC curve and the PR curve.

3.2.1 Area Under ROC Curve (AUROC)

In machine learning problems with binary outcome, the ROC curve is very effective to take the sensitivity and the specificity of the classifier into account by plotting the false positive rate (FPR) on the x-axis and the true positive rate (TPR) on the y-axis. AUC summarizes this plot by measuring the area between the line and the x-axis, which is the area under ROC curve (AUROC). Let us assume that \( M \) is the number of test samples, \( V \in [0, 1]^M \) contains the sorted PCVs of test samples in descending order, \( T \in [0, 1]^M \) and \( F \in [0, 1]^M \) contain the corresponding TPR and FPR values, respectively, where the threshold for entry \( i \) is set to \( V[i] \), and \( T[0] = F[0] = 0 \). In case that there is no tie in \( V \), the privacy-friendly AUROC computation is as follows:

\[
AUROC = \sum_{i=1}^{M} \left( T[i] \cdot (F[i] - F[i-1]) \right)
\]
We will benefit from this observation in the privacy preserving AUROC computation. There are some frameworks in the literature that also perform 3-party secure computation. The recently published SecureNN [Wagh et al., 2018] has shown successful results in terms of performance in private CNN training. We compared our secure

\[
AUROC = \sum_{i=0}^{[\xi]} \left( T[\xi[i] - 1] \cdot (F[\xi[i]] - F[\xi[i] - 1]) + (T[\xi[i] - T[\xi[i] - 1]] \cdot (F[\xi[i]] - F[\xi[i] - 1])) \right) / 2
\]

As Equation (2) indicates, one only needs TPR and FPR values on the points where the PCV changes to obtain the exact AUROC. We will benefit from this observation in the privacy preserving AUROC computation.

3.2.2 Area Under PR Curve (AUPR)

Another model evaluation metric for machine learning problems with binary outcome is the PR curve which plots recall on the x-axis and precision on the y-axis, and it is generally preferred over AUROC for scenarios with class imbalances. The AUC summarizes this plot by measuring the area under the PR curve (AUPR). Since both precision and recall can change at the same time even without a tie, we measure the area by using the Equation (2) where T being the precision and F being the recall.

4 Framework

In this section, we give the definitions of basic operations of the framework that we use in ppAURORA. Note that we include only the operations with novelty due to the page limit. One can refer to the Supplement for the other operations.

Selecting One of Two Secret Shared Values: Algorithm 1 performs 3-party computation to select one of two secret shared values based on the secret shared bit value (functionality \(F_{\text{MUX}}\)). At the beginning of the protocol, \(S_0\) and \(S_1\) hold \((\langle x \rangle_0, \langle y \rangle_0, \langle b \rangle_0)\) and \((\langle x \rangle_1, \langle y \rangle_1, \langle b \rangle_1)\), respectively. At the end of the secure computation, \(S_0\) and \(S_1\) hold the fresh shares of \(z = x - b(x - y)\). We utilize the randomized encoding of multiplication [Applebaum, 2017]. As shown in Equation (3), we need to multiply two values owned by different parties in the computation of \(\langle b \rangle_0 \langle \langle x \rangle_1 - \langle y \rangle_1 \rangle\) and \(\langle b \rangle_1 \langle \langle x \rangle_0 - \langle y \rangle_0 \rangle\). We assume that \(S_2\) is the computation party and performs these multiplications via the randomized encoding.

\[
z = x - b(x - y)
= \langle x \rangle_0 + \langle x \rangle_1 - \langle b \rangle_0 \langle \langle x \rangle_0 - \langle y \rangle_0 \rangle - \langle b \rangle_1 \langle \langle x \rangle_1 - \langle y \rangle_1 \rangle - \langle b \rangle_0 \langle \langle x \rangle_1 - \langle y \rangle_1 \rangle - \langle b \rangle_1 \langle \langle x \rangle_0 - \langle y \rangle_0 \rangle
\]

Modulus Conversion: Algorithm 2 describes our 3-party protocol realizing the functionality \(F_{\text{MC}}\) that converts shares over \(\mathbb{Z}_K\) to fresh shares over \(\mathbb{Z}_L\) where \(L = 2K\). Assuming that \(S_0\) and \(S_1\) have the shares \(\langle x \rangle^K_0\) and \(\langle x \rangle^K_1\), respectively, the first step for \(S_0\) and \(S_1\) is to mask their shares by using the shares of the random value \(r \in \mathbb{Z}_K\) sent by \(S_2\). Afterwards, they reconstruct \(x + r \in \mathbb{Z}_K\), by first computing \(\langle y \rangle^K_i = \langle x \rangle^K_i + \langle r \rangle^K_i\) for \(i \in \{0, 1\}\) and sending these values to each other. Along with the shares of \(r \in \mathbb{Z}_K\), \(S_2\) also sends the information in boolean shares telling whether the summation of the shares of \(r\) wraps so that \(S_0\) and \(S_1\) can convert \(r\) from the ring \(\mathbb{Z}_K\) to the ring \(\mathbb{Z}_L\). Once they reconstruct \(y \in \mathbb{Z}_K\), \(S_0\) and \(S_1\) can change the ring of \(y\) to \(\mathbb{Z}_L\) by adding \(K\) to one of the shares of \(y\) if \(\langle y \rangle^K_0 + \langle y \rangle^K_1\) wraps. After conversion, the important detail regarding \(y \in \mathbb{Z}_L\) is to fix the value of it. In case \((x + r) \in \mathbb{Z}_K\) wraps, which we identify by using PC, depending on the boolean share of the outcome of PC, \(S_0\) or \(S_1\) or both add \(K\) to their shares. If both adds, this means that there is no addition to the value of \(y \in \mathbb{Z}_L\). At the end, \(S_1\) subtracts \(r_i \in \mathbb{Z}_L\) from \(y_i \in \mathbb{Z}_L\) and obtains \(x_i \in \mathbb{Z}_L\) for \(i \in \{0, 1\}\).

Comparison of Two Secret Shared Values: Algorithm 3 gives the definition of the 3-party protocol for the functionality \(F_{\text{CMP}}\) that compares two secret shared values \(x\) and \(y\), and outputs zero if \(x \geq y\), 1 otherwise. In Algorithm 3, we find the value of \((x - y)^2 - 1\) that indicates the most significant bit (MSB) of \((x - y)\). First, \(S_i\) where \(i \in \{0, 1\}\) computes \(\langle d \rangle_i^K = (\langle x \rangle^K_i - \langle y \rangle^K_i) \mod K\). \(S_i\) converts the ring of \(d\) from \(\mathbb{Z}_K\) to \(\mathbb{Z}_L\) by calling MC. Finally, \(S_0\) and \(S_1\) subtract the shares of \(d\) over \(\mathbb{Z}_L\) from the shares of \(x - y\) and map the result to 1 if it equals \(K\), otherwise 0.

Division: Algorithm 4 gives the definition of the 3-party protocol realizing the functionality \(F_{\text{DIV}}\) that computes \(x/y\). \(S_0\) and \(S_1\) hold shares of \(x\) and \(y\), and DIV outputs fresh shares of \(x/y\). DIV computes \(x/y\) correctly when \(S_0\) and \(S_1\) know the upper bound for \(x\) and \(y\). Thus, it is not a general purpose method over \(\mathbb{Z}_L\). We rather specifically designed it for ppAURORA since \(S_0\) and \(S_1\) know the upper bound of inputs to DIV in the computation of the AUC.

4.1 Complexities of Our Protocols

There are some frameworks in the literature that also perform 3-party secure computation. The recently published SecureNN [Wagh et al., 2018] has shown successful results in terms of performance in private CNN training. We compared our secure
protocols with the corresponding building blocks from SecureNN that one needs to use for computing AUC. Table I summarizes the comparison. It is clear that our protocols have lower costs. Therefore, we can deduce that our framework outperforms SecureNN in AUC computation. Our novel protocols (MUX, MC, CMP, DIV) require at most 5 communication rounds, which is the most important factor affecting the performance of secure computing protocols. Our new protocols show an improvement in terms of round complexities.

**Table 1: Complexity comparison of distinguishing protocols of ppAURORA with SecureNN Wagh et al. [2018]**

### 5 ppAURORA Computation

In this section, we give the description of our protocol for ppAURORA computation. In ppAURORA, we have data owners that outsource their PCVs and the ground truth labels in secret shared form and three non-colluding servers that perform 3-party computation on secret shared PCVs to compute AUC. The protocol starts with outsourcing by the data sources. Afterward, the servers perform the desired calculation privately. Finally, they send the shares of the result back to the data sources. The communication between all parties is performed over a secure channel (e.g., TLS).

**Algorithm 1: Select Share (MUX)**

**Algorithm 2: Modulus Conversion (MC)**

**Algorithm 3: Comparison of two secret shared values (CMP)**

**Algorithm 4: Division (DIV)**

**Outsourcing:** At the start of ppAURORA, each data owner $H_i$ has a list of PCVs and corresponding ground truth labels for $i \in \{1, \ldots, n\}$. Then, each data owner $H_i$ sorts its whole list $T_i$ according to PCVs in descending order and divides it into two additive shares $T_{i0}$ and $T_{i1}$, and sends $T_{i0}$ and $T_{i1}$ to $S_0$ and $S_1$, respectively. We refer to $S_0$ and $S_1$ as proxies.
Sorting: After the outsourcing phase, $S_0$ and $S_1$ obtain the shares of individually sorted lists of PCVs of the data owners. Afterwards, the proxies need to perform a merging operation on each pair of individually sorted lists and continue with the merged lists until they obtain the global sorted list of PCVs. This can be considered as the leaves of a binary tree merging into the root node, which is, in our case, the global sorted list. Due to the high complexity of privacy preserving sorting, we decided to make the sorting parametric to adjust the trade-off between privacy and practicality. Let $\delta = 2a + 1$ be this parameter that determines the number of PCVs that will be added to the global sorted list in each iteration for $a \in \mathbb{N}$, $T_{i1}$ and $T_{i2}$ be the shares of two individually sorted lists of PCVs in $S_{i}$ for $k \in \{0, 1\}$ and $|T_i| \geq |T_j|$ where $| \cdot |$ is size operator. At the beginning, the proxies privately compare the lists elementwise. They utilize the results of the comparison in MUXs to privately exchange the shares of PCVs in each pair if the PCV in $T_i$ is larger than the PCV in $T_j$. In the first MUX, they input the share in $T_{i1}$ to MUX first and then the share in $T_{i2}$ along with the share of the result of the comparison to select the larger of the PCVs and move it to $T_{i2}$. In the second MUX, they reverse the order to select the smaller of the PCVs and move it to $T_{i1}$. We call this stage shuffling. Then, they move the top PCV of $T_{i1}$ to the merged list of PCVs. If $\delta \neq 1$, then they continue comparing the top PCVs in the lists and moving the largest of them to the merged list. Once they move $\delta$ PCVs to the merged list, they shuffle the lists again. Until finishing up the PCVs in $T_{i1}$, the proxies follows shuffling-moving cycle.

The purpose of the shuffling is to increase the number of candidates for a specific position and, naturally, lower the chance of matching a PCV in the individually sorted lists to a PCV in the merged list. The highest possible chance of a matching is 50%. This result in a very low chance of guessing the matching of whole PCVs in the list. Regarding the effect of $\delta$ on the privacy, it is important to note that $\delta$ needs to be an odd number to make sure that shuffling always leads to increment in the number of candidates. Even value of $\delta$ may cause ineffective shuffling during the sorting. Furthermore, $\delta = 1$ provides the utmost privacy, which means that the chance of guessing the matching of the whole PCVs is 1 over the number of all possible merging of those two individually sorted lists. However, the execution time of sorting with $\delta = 1$ can be relatively high. For $\delta \neq 1$, the execution time can be low but the number of possible matching of PCVs in the individually sorted list to the merged list decreases in parallel to the increment of $\delta$. As a guideline on the choice of $\delta$, one can decide it based on how much privacy loss any matching could cause on the specific task. In case of $\delta \neq 1$ and $|T_{i1}| = 1$ at some point in the sorting, the sorting continues as if it had just started with $\delta = 1$ to make sure that the worst case scenario for guessing the matching can be secured. More details of the sorting phase are in the Supplement.

5.1 Secure Computation of AUROC

Once $S_0$ and $S_1$ obtain the global sorted list of PCVs, they calculate the AUROC based on this list by employing one of the versions of AUROC depending on whether there exists tie in the list.

```
input : \((T)_i = \{(\text{con}_1),(\text{label}_1)\},...,\{(\text{con}_N),(\text{label}_M)\}\),
\((T)_i\) is a share of the global sorted list of PCVs and labels
1 For each \(i \in \{0,1\}\), \(S_i\) executes Steps 2-11
2 \((T)_i\) \leftarrow 0, \((P)_i\) \leftarrow 0, \((pFP)_i\) \leftarrow 0, \((pTP)_i\) \leftarrow 0, \((N)_i\) \leftarrow 0
3 foreach item \((t)_i \in \mathcal{T}\) do
    4 \((T)_{i1}\) \leftarrow \((T)_{i2}\) + \((t.label)_i\)
    5 \((P)_{i1}\) \leftarrow \((P)_{i2}\) + 1
    6 \((pFP)_{i1}\) \leftarrow \((pFP)_{i2}\) - \((pFP)_{i1}\)
    7 \((pTP)_{i1}\) \leftarrow \((pTP)_{i2}\) - \((pTP)_{i1}\)
    8 \((N)_{i1}\) \leftarrow \((N)_{i2}\) + \((A)_{i1}\)
    9 \((A)_{i1}\) \leftarrow \text{MUL}(\((P)_{i1}\), \((pFP)_{i1}\))
    10 \((\text{ROC})_{i1}\) \leftarrow \text{DIV}(N)_{i1},(D)_{i1}\)
```

Algorithm 5: Secure AUROC computation without ties

5.1.1 Secure AUROC Computation without Ties

In Algorithm 5 we compute the AUROC as shown in Equation 1 because we assume that there is no tie in the sorted list of PCVs. At the end of the secure computation, the shares of numerator $N$ and denominator $D$ are computed where $AUROC = N/D$. $S_i$ for $i \in \{0, 1\}$ knows the number of test samples $M$. Thus $S_i$ can determine the upper bound for $N$ and $D$ and DIV can be used to calculate $AUROC = N/D$. With the help of high numeric value precision of the results, most of the machine learning algorithms yield different PCVs for samples. Therefore, such an approach to compute the AUROC is applicable to most of the machine learning tasks. However, in case of a tie between samples from two classes in the PCVs, it does not guarantee that it gives the exact AUROC. Depending on the order of the samples, it approximates the score. To have a more accurate AUROC, we will propose another version with a slightly higher communication cost in the next section.

```
\text{input : } (T)_i = \{(\text{con}_1),(\text{label}_1)\},...,\{(\text{con}_N),(\text{label}_M)\}
\text{ }(T)_i\text{ in a share of the global sorted list of PCVs and labels}
1 For each \(i \in \{0,1\}\), \(S_i\) executes Steps 2-18
2 \((T)_{i1}\) \leftarrow 0, \((P)_{i1}\) \leftarrow 0, \((pFP)_{i1}\) \leftarrow 0, \((pTP)_{i1}\) \leftarrow 0, \((N)_{i2}\) \leftarrow 0
3 foreach item \((t)_i \in \mathcal{T}\) do
    4 \((T)_{i1}\) \leftarrow \((T)_{i2}\) + \((t.label)_i\)
    5 \((P)_{i1}\) \leftarrow \((P)_{i2}\) + 1
    6 \((pFP)_{i1}\) \leftarrow \((pFP)_{i2}\) - \((pFP)_{i1}\)
    7 \((pTP)_{i1}\) \leftarrow \((pTP)_{i2}\) - \((pTP)_{i1}\)
    8 \((N)_{i1}\) \leftarrow \((N)_{i2}\) + \((A)_{i2}\)
    9 \((A)_{i2}\) \leftarrow \text{MUL}(\((A)_{i1}\), \((t.con)\))
    10 \((A)_{i1}\) \leftarrow \text{MUL}(\((P)_{i1}\), \((pFP)_{i1}\))
    11 \((A)_{i2}\) \leftarrow \text{MUL}(\((A)_{i1}\), \((t.con)\))
    12 \((A)_{i1}\) \leftarrow \text{MUL}(\((A)_{i1}\), \((t.con)\))
    13 \((A)_{i2}\) \leftarrow \text{MUL}(\((A)_{i1}\), \((t.con)\))
    14 \((\text{ROC})_{i1}\) \leftarrow \text{DIV}(\((N)_{i1}\), (D)_{i1}\))
```

Algorithm 6: Secure AUROC computation with tie
5.1.2 Secure AUROC Computation with Ties

To detect ties in the list of PCVs, $S_0$ and $S_1$ compute the difference between each PCV and its following PCV. $S_0$ computes the modular additive inverse of its shares. The proxies apply a common random permutation to the bits of each share in the list to prevent $S_1$ from learning the non-zero relative differences. They also permute the list of shares using a common random permutation to shuffle the order of the real test samples. Then, they send the list of shares to $S_2$. $S_2$ XORs two shares and maps the result to one if it is greater than zero and zero otherwise. Then, proxies privately map PCVs to zero if they equal to their previous PCV and one otherwise. This phase is depicted in Algorithm 7. In Algorithm 8, $S_0$ and $S_1$ use these mappings to take only the PCVs which are different from their subsequent PCV into account in the computation of the AUROC based on the Equation 2. In Algorithm 8, the DIV method is used because the upper limit for the numerator and denominator is known, as in the AUROC computation described in Section 5.1.1.

5.2 Secure AUPR Computation

As in the AUROC computation described in Section 5.1.2, $S_0$ and $S_1$ map a PCV in the global sorted list to zero if it equals to the previous PCV and one otherwise by running the Algorithm 7. Then, we use Equation 2 to calculate AUPR as shown in Algorithm 8. In the AUPR calculation, the denominator of each precision value is different. Therefore, we need to perform division in each iteration. Since the proxy servers can determine the upper bound for numerators and denominators, we can divide the AUROC operation to perform divisions.

Algorithm 7: Secure detection of ties

**Algorithm 8: Secure AUPR computation with tie**

6 Security Analysis

We provide all semi-honest simulation-based security proofs for the MUX, MC, CMP and DIV functions defined in our framework as well as the computations of ppAURORA in the Supplement.

7 Dataset

**DREAM Challenge Dataset:** To demonstrate the correctness of ppAURORA and its applicability to a real-life problem, we utilized data from the first subchallenge of the Acute Myeloid Leukemia (AML) outcome prediction challenge from the DREAM Challenges [Noren et al., 2016]. We chose the submission of the team with the lowest score in the leaderboard with accessible files, which is the team Snail. The training dataset has 191 samples, among which 136 patients have complete remission. The ground truth labels of the training samples can be reached at https://www.synapse.org/#!Synapse:syn2501858 and the submission of team Snail can be found at https://www.synapse.org/#!Synapse:syn2700200.

**Synthetic Dataset:** Since we aimed to analyze the scalability of ppAURORA at different settings, we generated a synthetic dataset with no restriction other than having the PCVs between 0 and 1.
Figure 1: (a) The execution time of various settings to evaluate the scalability of ppAURORA to the number of samples for a fixed number of parties and (b) to the number of parties for a fixed number of samples in each party. (c) The effect of $\delta$ on the execution time is shown.

8 Results

Experimental Setup: We conducted our experiments on Amazon EC2 t2.xlarge instances. For the LAN setting, we utilized only the instances in the Frankfurt region. For the WAN setting, we additionally selected one instance from both London and Paris.

Experiments on DREAM Challenge Dataset: We conducted the experiments on LAN setting. We assume that the result needs to be precise in the first three decimal places. Thus, we set the precision of the division operation to cover up to the fourth decimal place. To assess the correctness, we computed the AUROC by ppAURORA and compared it to the result obtained without privacy. Both settings gave $AUROC = 0.693$. To check the correctness of AUROC with no-tie of ppAURORA, we randomly picked one of the samples in tie condition and generated a subset of the samples with no tie. We obtained the same AUROC with no-tie version of AUROC of ppAURORA as the non-private computation. Additionally, we computed AUPR by employing ppAURORA and verified that both private and non-private gave $AUPR = 0.844$. These results indicate that without sacrificing the privacy of the data, ppAURORA can compute the exact same AUC as one could obtain on the pooled test samples. To justify the collaborative evaluation, we performed 1000 repetitions of the non-private AUROC computation with $N \in \{5, 10, 20, 40, 80, 160\}$ samples, which are randomly drawn from the whole data. The AUC starts to become more and more stable when we increase the size of the samples.

Experiments on Synthetic Dataset: We evaluated no-tie and with-tie versions of AUROC and AUPR of ppAURORA with $\delta = 1$ on the settings in which the number of data sources is 16 and the number of samples is $N \in \{64, 125, 250, 500, 1000\}$. The results showed that ppAURORA scales almost quadratically in terms of both communication costs among all parties and the execution time of the computation. Figure 1a displays the results. We also analyzed the performance of all computations of ppAURORA on a varying number of data sources. We fixed $\delta = 1$, the number of samples in each data source to 1000 and experimented with $D$ data sources where $D \in \{2, 4, 8, 16\}$. Similar to the previous analysis, ppAURORA scales around quadratically to the number of data sources and Figure 1b summarizes the results. We also analyzed the effect of $\delta \in \{1, 3, 5, 11, 25, 51, 101\}$ by fixing the number of data sources to $D$ and the number of samples in each data source to 1000. The execution time shown in Figure 1c displays logarithmic decrease for increasing $\delta$. In all analyses, since the dominating factor is sorting, the execution times of the computations are close to each other. Additionally, our analysis showed that LAN is 12 to 14 times faster than WAN on average due to the high round trip time of WAN, which is approximately 13.2 ms. However, even with such a scaling factor, ppAURORA can be deployed in real life scenarios if the alternative is a more time-consuming approval process required for gathering all data in one place still protecting the privacy of data. We provided the detailed results as tables in the Supplement.

9 Conclusion

In this work, we presented a novel secure 3-party computation framework and its application, ppAURORA, to compute AUC of the ROC and PR curves privately even when there exist ties in the PCVs. We proposed four novel protocols in the framework, which are MUX to select one of two secret shared values, MC to convert the ring of a secret value from $2^{\ell-1}$ to $2^\ell$, CMP to compare two secret shared values and DIV to divide two secret shared values. They have low round and communication complexities. The framework and its application ppAURORA are secure against passive adversaries in the honest majority setting. We implemented ppAURORA in C++ and demonstrated that ppAURORA can both compute correctly and privately, and scales quadratically to the number of parties and the number of samples. To the best of our knowledge, ppAURORA is the first method that enables computing the exact AUC (AUROC and AUPR) privately and securely.
10 Supplementary Material

10.1 Framework

In this section, we give the definitions of basic operations that we utilized in addition to the novel operations given in the main paper.

10.1.1 Multiplication

In two-party setting, multiplication is performed using a multiplication triple [Beaver 1991] which is generated via homomorphic encryption or oblivious transfer. In our 3-party setting, $S_2$ generates the multiplication triple and sends the shares of it to $S_0$ and $S_1$. $S_0$ and $S_1$ hold $(\langle x \rangle_0, \langle y \rangle_0)$ and $(\langle x \rangle_1, \langle y \rangle_1)$, respectively, and secure multiplication outputs fresh shares of $xy$ (functionality $\mathcal{F}_{\text{MUL}}$).

1 Algorithm MUL()

input $: S_0$ and $S_1$ hold $(\langle x \rangle_0, \langle y \rangle_0)$ and $(\langle x \rangle_1, \langle y \rangle_1)$, respectively.
output $: S_0$ and $S_1$ get $\langle z \rangle_1$, respectively, where $z = x \cdot y$.
$S_2$ picks three random numbers $a$, $b$ and $c$ where $c = a \cdot b$
For each $i \in \{0, 1\}$, $S_i$ executes Steps 5-7
$S_i$ computes $\langle e \rangle_i = \langle x \rangle_i - \langle a \rangle_i$ and $\langle f \rangle_i = \langle y \rangle_i - \langle b \rangle_i$
$S_i$ reconstructs $e$ and $f$ by exchanging the shares with $S_{1-i}$
$S_i$ computes $\langle z \rangle_i = i \cdot e \cdot f + f \cdot \langle x \rangle_i + e \cdot \langle y \rangle_i + \langle c \rangle_i$

Algorithm 9: Multiplication

10.1.2 Comparison of a Secret Shared Value and a Plain Value

In this function, a value $r$ in the ring $\mathbb{Z}_K$ whose bits are secret shared to $S_0$ and $S_1$ in the ring $P$ where $P = 67$ is compared with a common value $y$. At the end of the secure computation, $S_2$ learns a bit $n' = n + (r > y)$. We get the definition of this functionality $\mathcal{F}_{\text{PC}}$ from [Wagh et al. 2018] where it is called as Private Compare (PC). The only change that we made is $S_2$ sends the fresh boolean shares of $n'$ to $S_1$ and $S_2$. PC is described in Algorithm 10 in the Supplement.

10.2 Result Tables

In this section, we provide the detailed results of the experiments with ppAURORA to compute AUROC and AUPR in Tables 2 and 3 respectively.

10.3 AUC Stability Analysis

We analyzed the stability of the AUROC based on the number of test samples to justify the collaborative evaluation. We experimented with $N \in \{5, 10, 20, 40, 80, 160\}$ test samples which are randomly chosen from the DREAM Challenge Dataset. In order to have a fair evaluation, we repeated these experiments 1000 times. The experiments showed that the AUROC becomes more reliable and stable if we increase the number of test samples. In case a data source has no more additional test samples, the collaborative evaluation can be a best option. Figure 2 summarizes the results of the analysis.

10.4 Privacy Preserving Merging

We include some sorting examples to demonstrate the process. In Figure 3, we show the merging of two lists of PCVs with the same size. In this example, $\delta = 1$. By this setting, we do not decrease the number of possible merging of two individually sorted lists, which can be computed as:
$$\sum_{i=0}^{\left|L_1\right|-1} \binom{\left|L_1\right|+1}{i+1} \binom{\left|L_2\right|-1}{i}$$
(4)

where $L_1$ and $L_2$ are the individually sorted lists and $\left|\cdot\right|$ denotes the size operation.

Since the cost of having fully private merging is high and may not necessary for some applications, we can have an intermediate solution. By setting $\delta$ to a higher odd value, we can speed up the merging process. Figure 4 and 5 demonstrates an example of such a merging with $\delta = 3$. The total number of shuffling is $\delta$ and 3, respectively.
Table 2: The summary of the results of the experiments with ppAURORA to compute AUROC with and without tie on synthetic data. The left side of ‘/acute.ts1/acute.ts1’ represents without-tie results and the right side of it represents with-tie results. M represents the number of data sources and N represents the number of samples in one data sources. UNB represents the unbalanced sample distribution, which is {12, 18, 32, 58, 107, 258, 507, 1008}. 

Algorithm 10: Private Compare [Wagh et al. 2018]
| M × N  | δ   | $P_1$     | $P_2$     | Helper | Total       | Time (sec) |
|--------|-----|-----------|-----------|--------|-------------|------------|
| 16 × 64| 1   | 73.79     | 73.75     | 139.70 | 287.24      | 15.49      |
| 16 × 125| 1   | 280.25    | 280.17    | 530.90 | 1091.32     | 39.50      |
| 16 × 250| 1   | 1118.39   | 1118.24   | 2120.17| 4356.80     | 160.25     |
| 16 × 500| 1   | 4468.38   | 4468.08   | 8470.66| 17407.12    | 666.28     |
| 16 × 10³| 1   | 17863.16  | 17862.55  | 33874.20| 69599.90    | 2639.34    |
| 2 × 10³| 1   | 149.58    | 149.55    | 283.02 | 582.14      | 23.09      |
| 4 × 10³| 1   | 894.58    | 894.49    | 1695.60| 3484.67     | 137.44     |
| 8 × 10³| 1   | 4170.19   | 4169.94   | 7905.92| 16246.05    | 635.72     |
| 8 × 0³  | 3   | 2070.93   | 2070.73   | 3924.48| 8066.14     | 346.11     |
| 8 × 0³  | 5   | 1385.75   | 1385.57   | 2624.46| 5395.77     | 249.01     |
| 8 × 0³  | 11  | 695.77    | 695.61    | 1316.72| 2708.10     | 152.96     |
| 8 × 0³  | 25  | 324.68    | 324.52    | 613.25 | 1262.45     | 104.86     |
| 8 × 0³  | 51  | 164.75    | 164.59    | 309.92 | 639.25      | 85.14      |
| 8 × 0³  | 101 | 87.73     | 87.58     | 163.67 | 338.98      | 75.98      |
| 8 × 250 | 1   | 261.49    | 261.43    | 495.38 | 1018.29     | 35.95      |
| 8 × UNB | 1   | 332.03    | 331.94    | 629.20 | 1293.17     | 46.00      |

Table 3: The summary of the results of the experiments of AUPR computation with ppAURORA on synthetic data. $M$ represents the number of data sources and $N$ represents the number of samples in one data sources. $UNB$ represents the unbalanced sample distribution, which is $\{12, 18, 32, 58, 107, 258, 507, 1008\}$.

In Figure 6, we show the sorting when the number of PCVs in the second list is 1 to justify why we set $\delta = 1$ regardless of the initial $\delta$ due to the privacy reasons. By setting $\delta$, we secure that the number of possible PCVs in each position of the global sorted list is the same as one can see on such a merging without privacy. The beginning and the end of the global sorted list have two possible matching, however, due to the nature of the individual sorting, the number of possible PCVs for the other positions is only 3, which is lower than the ones in Figure 3.

10.5 Security Analysis

Here, we provide semi-honest simulation-based security proofs for the MUX, MC, CMP and DIV functions we have defined in our framework. Since the protocols we propose for AUC calculation use MUX, MC, CMP, DIV and previously defined functions, we prove the security of the main protocols in the F-hybrid model by proving the security of each function we call.

**Lemma 10.1.** The protocol MUL in Algorithm 9 in the Supplement securely realizes the functionality $\mathcal{F}_{\text{MUL}}$. 

In Figure 2, we show the AUROC for varying number of test samples randomly chosen from the whole dataset.
We prove the security of our protocol. During the protocol execution, we first prove the correctness of our protocol.

**Proof.** The protocol sees \( S \) receive any values. Thus the view of \( S \) is empty and it is very easy to prove security in case \( S \) is corrupted. \( S \) where \( i \in \{0,1\} \) sees \( \langle a \rangle_i, \langle b \rangle_i, \langle c \rangle_i, \langle e \rangle_i \) and \( \langle f \rangle_i \). These values are uniformly distributed random values and hence can be perfectly simulated.

\[
\langle z \rangle_0 + \langle z \rangle_1 = xy.
\]

We prove the security of our protocol. During the protocol execution, \( S_2 \) sends a multiplication triple to \( S_0 \) and \( S_1 \) and does not receive any values. Thus the view of \( S_2 \) is empty and it is very easy to prove security in case \( S_2 \) is corrupted. \( S_i \) where \( i \in \{0,1\} \) sees \( \langle a \rangle_i, \langle b \rangle_i, \langle c \rangle_i, \langle e \rangle_i \) and \( \langle f \rangle_i \). These values are uniformly distributed random values and hence can be perfectly simulated.

\[
\langle z \rangle_0 + \langle z \rangle_1 = x - (b)x + y.
\]

**Lemma 10.2.** The protocol MUX in Algorithm[4] in the main paper securely realizes the functionality \( F_{\text{MUX}} \).

**Proof.** We first prove the correctness of our protocol. \( \langle z \rangle_i \) is the output of \( S_i \) where \( i \in \{0,1\} \). We need to prove that \( \text{Reconstruct}(\langle z \rangle_i) = (1 - b)x + by \).

\[
\langle z \rangle_0 + \langle z \rangle_1 = x - (b)x + y.
\]
we cannot fix the summation value \( y \). At the beginning, \( S \) shares of the inputs and any value learned by \( S \) is perfectly simulatable. For each \( r \) uniformly random because it contains uniformly random values \( r \) they are generated using uniformly random values \( r \). Next we prove the security of our protocol.

**Lemma 10.3.** \( \text{The protocol PC in Algorithm 10 in the Supplement securely realizes the functionality } \mathcal{F}_{PC}. \)

**Proof.** The proof of PC is given in [Wagh et al. 2018].

**Lemma 10.4.** \( \text{The protocol MC in Algorithm 2 in the main paper securely realizes the functionality } \mathcal{F}_{MC} \text{ in } \mathcal{F}_{PC} \text{ hybrid model.} \)

**Proof.** First, we prove the correctness of our protocol by showing \((\langle x \rangle_0^K + \langle x \rangle_1^K) \mod K = (\langle x \rangle_0 + \langle x \rangle_1) \mod L. \) In the protocol, \( y = (x + r) \mod K \) and isWrap\( (x, r, K) = r > y, \text{ that is, } \text{isWrap}(x, r, K) = 1 \text{ if } r > y, 0 \text{ otherwise.} \)

At the beginning, \( S_0, S_1 \) and \( S_2 \) call \( \mathcal{F}_{PC} \) to compute \( c = r > y \) and \( S_0 \) and \( S_1 \) obtain the boolean shares \( e_0 \) and \( e_1 \), respectively. Besides, \( S_2 \) sends also the boolean shares \( w_0 \) and \( w_1 \) of \( w = \text{isWrap}(\langle r \rangle_0, \langle r \rangle_1, K) \) to \( S_0 \) and \( S_1 \), respectively. If \( \text{isWrap}(\langle y \rangle_0, \langle y \rangle_1, K) = 1 \) then \( S_0 \) adds \( K \) to \( \langle y \rangle_0 \) to change the ring of \( y \) from \( K \) to \( L \). To convert \( r \) from ring \( K \) to ring \( L \), \( S_0 \) and \( S_1 \) add \( K \) to their shares of \( r \) based on their boolean shares \( w_0 \) and \( w_1 \), respectively. If \( w_0 = 1 \), then \( S_0 \) adds \( K \) to its \( r_1 \) and \( S_1 \) does the similar with its shares. Later, we need to fix is the summation of \( x \) and \( r \), that is the value \( y \). In case of \( x + r \geq K \), we cannot fix the summation value \( y \) in ring \( L \) by simply converting it from ring \( K \) to ring \( L \). This summation should be \( x + r \) in ring \( L \) rather than \( (x + r) \mod K \). To handle this problem, \( S_0 \) and \( S_1 \) add \( K \) to their shares of \( y \) based on their shares \( e_0 \) and \( e_1 \). As a result, we convert the values \( y \) and \( r \) to ring \( L \) and fix the value of \( y \) if necessary. The final step to obtain \( x_i \) for party \( S_i \) is simply subtract \( r_i \) from \( y_i \) where \( i \in \{0, 1\} \).

Next, we prove the security of our protocol. \( S_2 \) involves this protocol in execution of \( \mathcal{F}_{PC} \). We give the proof \( \mathcal{F}_{PC} \) above. At the end of the execution of \( \mathcal{F}_{PC} \), \( S_2 \) learns \( n' \). However, \( n' = n \oplus (x > r) \) and \( S_2 \) does not know \( n \). Thus \( n' \) is uniformly distributed and can be perfectly simulated with randomly generated values. \( S_i \) where \( i \in \{0, 1\} \) sees fresh shares of \( \langle r \rangle_i^K, \langle r_i[j] \rangle_i^B \) \( \forall j \in [\ell], w_i^B \) and \( n_i^B \). These values can be perfectly simulated with randomly generated values.

---

**Figure 4:** The merging of two same size lists with \( \delta = 3 \). We again start with shuffling and then move the top PCV of the first list into the global sorted list. Afterward, we compare the second element of the first list and the top element of the second list via PC. The proxies reconstruct the result of this comparison and move the share of the larger of the compared PCVs to the global sorted list. They continue until they move \( \delta \) PCVs to the global sorted list. Then, they shuffle and repeat the same procedure until there is no PCV in the first list.
Figure 5: The merging of two same size lists with $\delta = 3$. This example shows how the merging happens in case all the PCVs are taken from the first list.

Figure 6: The merging of two lists with $\delta = 1$. The second list has only one PCV. The red arrows represent shuffling, the black ones denote moving the larger of the first list to the global sorted list. The progress of the global sorted list is shown by the grey arrows. Each color in the boxes represents different number of candidate PCVs. The coding of the colors are shown on the right most side of the figure.

Lemma 10.5. The protocol CMP in Algorithm 3 in the main paper securely realizes the functionality $F_{CMP}$ in $F_{MC}$ hybrid model.

Proof. First, we prove the correctness of our protocol. Assume that we have $\ell$-bit number $u$. $v = u - (u \mod 2^{\ell-1})$ is either 0 or $2^{\ell-1}$. $v$ is only represented with the most significant bit (MSB) of $u$. In our protocol, $\langle z \rangle_i$ is the output of $S_i$ where $i \in \{0, 1\}$. We need to prove that $\text{Reconstruct}(\langle z \rangle_i)$ is 1 if $x < y$ and 0 otherwise. $S_i$ where $i \in \{0, 1\}$ computes $d_i^K = (x_i - y_i) \mod K$ which is a share of $d$ over $K$. $S_i$ computes $d_i$ which is a share of $d$ over $L$ by invoking MC. Note that $z = x - y - \text{Reconstruct}(\langle d \rangle_i)$ and all bits of $z$ are 0 except MSB of $z$ which is equal to MSB of $(x - y)$. Now we need to map
z to 1 if it's equal to K and 0 if it's equal to 0. S0 sends the z0 and z0 + K in random order to S2 and S1 sends the z1 to S2. S2 reconstructs two different values, divides these values by K, creates two additive shares of them, and sends these shares to S0 and S1. Since S0 and S1 know the order of the real MSB value, they correctly select the shares of its mapped value.

Second, we prove the security of our protocol. Si where i ∈ {0, 1} sees (d), which is a fresh share of d and ⟨a[0]⟩, and ⟨a[1]⟩, one of them is a fresh share of the MSB of x − y and the other is a fresh share of the complement of the MSB of x − y. Thus the view of Si can be perfectly simulated with randomly generated values.

Lemma 10.6. The protocol DIV in Algorithm[4] in the main paper securely realizes the functionality FDIV.

Proof. We first prove the correctness of our protocol. ⟨z⟩i is the output of Si where i ∈ {0, 1}. We prove that

\[
\text{Reconstruct}(⟨z⟩_i) = \frac{r_F}{y}.
\]

\[
⟨z⟩_0 + ⟨z⟩_1 = \frac{(r_1(x)_0 + r_0(y)_0 + r_1(x)_1 + r_0(y)_1)F}{r_1(y)_0 + r_1(y)_1} - \frac{r_0F}{r_1}
\]

\[
= \frac{(r_1y + r_0y)}{y} - \frac{r_0F}{r_1}
\]

\[
= \frac{x}{y} - \frac{r_0F}{r_1}
\]

DIV method produces correct results when S0 and S1 know the upper limit of x and y values. In this case, the values of r0 and r1 are chosen in such a way that the values r1⟨x⟩0 + r0⟨y⟩0 + r1⟨x⟩1 + r0⟨y⟩1 and r1⟨y⟩0 + r1⟨y⟩1 do not wrap around ZL. If wrapping occurs around ZL, DIV method produces the wrong result.

Next we prove the security of our protocol. S2 gets a1, b0 and b1. All these values are uniformly random values because they are generated using uniformly random values r0 and r1. As a result, any value learned by S2 is perfectly simulated. For each i ∈ {0, 1}, Si learns a fresh share of the output. Thus S1 cannot associate the share of the output with the shares of the inputs and any value learned by Si is perfectly simulatable.

Lemma 10.7. The protocol in Algorithm[5] in the main paper securely computes AUROC in (F_MUL, F_DIV) hybrid model.

Proof. In the protocol, we separately calculate the numerator N and the denominator D of the AUROC, which can be expressed as \(\text{AUROC} = \frac{N}{D}\). Let us first focus on the computation of D. It is equal to the multiplication of the number of samples with label 1 by the number of samples with label 0. In the end, we have the number of samples with label 1 in TP and calculate the number of samples with label 0 by \(FP - TP\). Then, the computation of D is simply the multiplication of these two values. In order to compute N, we employed Equation[1] in the main paper. We have already shown the denominator part of it. For the numerator part, we need to multiply the current TP by the change in FP and sum up these multiplication results. \(⟨A⟩ ← \text{MUL(⟨TP⟩, (⟨FP⟩ − ⟨pFP⟩))}\) computes the contribution of the current sample on the denominator and we accumulate all the contributions in N, which is the numerator part of Equation[1] in the main paper. Therefore, we can conclude that we correctly compute the AUROC.

Next, we prove the security of our protocol. S1 where i ∈ {0, 1} sees \(⟨RL⟩\) j ∈ M, \(⟨A⟩\) j ∈ M, ⟨D⟩ and ⟨ROC⟩ which are fresh shares of these values. Thus the view of S1 is perfectly simulatable with uniformly random values.

Lemma 10.8. The protocol in Algorithm[7] in the main paper securely marks the location of ties in the list of prediction confidences.

Proof. For the correctness of our protocol, we need to prove that for each index j in T, \(t[j].\text{con} = 1\) if \((C[j] − C[j + 1]) = 0\), \(t[j].\text{con} = 1\), otherwise. We first calculate the difference of successive items in C. Let assume we have two additive shares \(⟨⟨a⟩⟩_0, 0⟩ \) of a over the ring ZL. If a = 0, then \(L − ⟨⟨a⟩⟩_0, 0⟩ = 0\) and if a ≠ 0, then \(L − ⟨⟨a⟩⟩_0, 0⟩ = 0\) where \(L − ⟨⟨a⟩⟩_0, 0⟩\) is the additive modular inverse of \(⟨⟨a⟩⟩_0, 0⟩\). We use this fact in our protocol. S0 computes the additive inverse of each item \(⟨⟨c⟩⟩_0 \) in \(⟨⟨C⟩⟩_0\) which is denoted by \(c⟩_0\), XORes \(c⟩_0\) with a common random number in R, which is denoted by \(c⟩_0′\) and permutes the bits of \(c⟩_0′\) with a common permutation \(σ\) which is denoted by \(c⟩_0′′\). S1 XORes each item \(c⟩_1 \) in \(⟨⟨C⟩⟩_1\) with a common random number in R which is denoted by \(c⟩_1′\) and permutes the bits of \(c⟩_1′\) with a common permutation \(σ\) which is denoted by \(c⟩_1″\). S1 XORes each item \(c⟩_1 \) in \(⟨⟨C⟩⟩_1\) with a common random number in R which is denoted by \(c⟩_1′\) and permutes the bits of \(c⟩_1′\) with a common permutation \(σ\) which is denoted by \(c⟩_1″\). S1 XORes each item \(c⟩_1 \) in \(⟨⟨C⟩⟩_1\) with a common random number in R which is denoted by \(c⟩_1′\) and permutes the bits of \(c⟩_1′\) with a common permutation \(σ\) which is denoted by \(c⟩_1″\). After receiving \(⟨D⟩_0\) and \(⟨D⟩_1\), S2 maps each item d of D to 0 if \(d⟩_0′ ⊕ ⟨d⟩_1 = 0\) which means \(d⟩_0 + ⟨d⟩_1 = 0\) and maps 1
if $\langle d \rangle_0 \oplus \langle d \rangle_1 \neq 0$ which means $\langle d \rangle_0 + \langle d \rangle_1 \neq 0$. After receiving a new share of $D$ from $S_2$, $S_i \in \{0,1\}$ removes dummy values and permutes remaining values by $\pi'$. Therefore, our protocol correctly maps items of $C$ to 0 or 1.

We next prove the security of our protocol. $S_i$ where $i \in \{0,1\}$ calculates the difference of successive prediction values. The view of $S_2$ is $D$ which includes real and dummy zero values. $S_i$ XORes each item of $\langle C \rangle_i$ with fresh boolean shares of zero, applies a random permutation to bits of each item of $C_i$, applies a random permutation $\pi$ on the reconstructed output of operations and $\pi$ outputs fresh shares of comparison of corresponding values in shuffling and selection operations. This ensures that our merge sort algorithm that $L_i$ is removed based on the result of $L_1$ element of corresponding values in the two lists. After the shuffling process, we know that the largest element of the two lists is the top element of $L_i$. Thus $S_2$ can not infer anything about real values in $C$. Furthermore, the number of repeating predictions is not known to $S_2$ due to a random permutation $\pi$.

Lemma 10.9. The protocol in Algorithm[6] in the main paper securely computes AUROC in $(\mathcal{F}_{MUL}, \mathcal{F}_{MUX}, \mathcal{F}_{DIV})$ hybrid model.

Proof. In order to compute the AUROC in case of tie, we utilize Equation[2] in the main paper, of which we calculate the numerator and the denominator separately. The calculation of the denominator $D$ is the same as Lemma[10.7]. The computation of the numerator $N$ has two different components, which are $N_1$ and $N_2$. $N_1$, more precisely the numerator of $(T[i-1] \star (F[i] - F[i-1]))$, is similar to no-tie version of privacy preserving AUROC computation. This part corresponds to the rectangle areas in ROC curve. The decision of adding this area $A$ to the cumulative area $N_1$ is made based on the result of the multiplication of $A$ by $t_{con}$. $t_{con} = 1$ indicates if the sample is one of the points of prediction confidence change, 0 otherwise. If it is 0, then $A$ becomes 0 and there is no contribution into $N_1$. If it is 1, then we add $A$ to $N_1$. On the other hand, $N_2$, which is the numerator of $(T[i] - T[i-1]) \star (F[i] - F[i-1])$, accumulates the triangular areas. We compute the possible contribution of the current sample to $N_2$. In case this sample is not one of the points that the prediction confidence changes, which is determined by $t_{con}$, then the value of $A$ is set to 0. If it is, then $A$ remains the same. Finally, $A$ is added to $N_2$. Since there is a division by 2 in the second part of Equation[2] in the main paper, we multiply $N_1$ by 2 to make them have common denominator. Afterwards, we sum $N_1$ and $N_2$ to obtain $N$. In order to have the term 2 in the common denominator, we multiply $D$ by 2. As a result, we correctly compute the numerator and the denominator of the AUROC.

Next, we prove the security of our protocol. $S_i$ where $i \in \{0,1\}$ sees $\{(RL)\}_{j \in M}, \{(A)\}_{j \in M}, \{(pFP)\}_{j \in M}, \{(pTP)\}_{j \in M}, \langle D \rangle$ and $\langle ROC \rangle$ which are fresh shares of these values. Thus the view of $S_i$ is perfectly simulatable with uniformly random values.

Lemma 10.10. The protocol in Algorithm[8] in the main paper securely computes AUPR in $(\mathcal{F}_{MUL}, \mathcal{F}_{MUX}, \mathcal{F}_{DIV})$ hybrid model.

Proof. In order to compute the AUPR, we utilize Equation[2] in the main paper of which we calculate the numerator and the denominator separately. We nearly perform the same computation with the AUROC computation in case of tie. The main difference is that we need perform division to calculate each precision value because denominators of each precision value are different. The rest of the computation is the same with the computation in Algorithm[6] in the main paper. The readers can follow the proof of Lemma[10.9].

Next, we prove the security of our protocol. $S_i$ where $i \in \{0,1\}$ sees $\{(RL)\}_{j \in M}, \{(T_{-PC})\}_{j \in M}, \{(A)\}_{j \in M}, \{(pPC)\}_{j \in M}, \{(pRC)\}_{j \in M}, \langle D \rangle$ and $\langle ROC \rangle$ which are fresh shares of these values. Thus the view of $S_i$ is perfectly simulatable with uniformly random values.

Lemma 10.11. The sorting protocol in Section 5 in the main paper securely merges two sorted lists in $(\mathcal{F}_{CMP}, \mathcal{F}_{MUX})$ hybrid model.

Proof. First, we prove the correctness of our merge sort algorithm. $L_1$ and $L_2$ are two sorted lists. In the merging of $L_1$ and $L_2$, the corresponding values are first compared using the secure CMP operation. The larger values are placed in $L_1$ and the smaller values are placed in $L_2$, after the secure MUX operation is called twice. This process is called shuffling because it shuffles the corresponding values in the two lists. After the shuffling process, we know that the largest element of the two lists is the top element of $L_1$. Therefore, it is removed and added to the global sorted list $L_3$. On the next step, the top elements of $L_1$ and $L_2$ are compared with the CMP method. The comparison result is reconstructed by $S_0$ and $S_1$ and the top element of $L_1$ or $L_2$ is removed based on the result of CMP and added to $L_3$. The selection operation also gives the largest element of $L_1$ and $L_2$ because $L_1$ and $L_2$ are sorted and the selection operation selects the larger of the top elements of $L_1$ and $L_2$. We show that shuffling and selection operations give the largest element of two sorted lists. This ensures that our merge sort algorithm that only uses these operations correctly merges two sorted lists in ordered manner.

Next, we prove the security of our merge sort algorithm. In the shuffling operation, CMP and MUX operations are called. CMP outputs fresh shares of comparison of corresponding values in $L_1$ and $L_2$. Shares of these comparison results are used in MUX operations and MUX operation generates fresh shares of the corresponding values. Therefore, $S_0$ and $S_1$ cannot precisely map these values to the values in $L_1$ and $L_2$. In the selection operation, CMP is called and the selection is performed based on the reconstructed output of CMP. $S_0$ and $S_1$ are still unable to map the values added to $L_3$ to the values in $L_1$ and $L_2$.\]
precisely because at least one shuffling operation took place before these repeated selection operations. Shuffling and \( \delta - 1 \) selection operations are performed repeatedly until the \( L_1 \) is empty. After each shuffling operation, the fresh share of the larger corresponding values in \( L_1 \) and the fresh share of the smaller corresponding values in \( L_2 \) are stored. The view of \( S_0 \) and \( S_1 \) are perfectly simulatable with random values due to the shuffling process performed at regular intervals.

It is possible in some cases to use unshuffled values in selection operations. To prevent this, the following rules are followed in the execution of the merge protocol. If there are two lists that do not have the same length, the longer list is chosen as \( L_1 \). If the \( \delta \) is greater than the length of the \( L_2 \) list, it is set to the largest odd value smaller or equal to the length of \( L_2 \) so that the unshuffled values that \( L_1 \) may have are not used in selection processes. If the length of \( L_2 \) is reduced to 1 at some point in the sorting, the \( \delta \) is set to 1. Thus \( L_2 \) will have 1 element until the end of the merge and shuffling is done before each selection.

10.5.1 Privacy against Malicious Adversaries

Araki et al. [2016] defined the notion of privacy against malicious adversaries in the client-server setting. In this setting, the servers performing secure computation on the shares of the inputs to produce the shares of the outputs do not see the plain inputs and outputs of the clients. This notion of privacy says that a malicious party cannot break the privacy of input and output of the honest parties. This setting is very similar to our setting. In our framework, two parties exchange a seed which is used to generate common randoms between them. Two parties randomize their shares using these random values which are not known to the third party. It is very easy to add fresh shares of zero to outputs of two parties with common random values shared between them. In our algorithms, we do not state the randomization of outputs with fresh shares of zero. Thus, our framework provides privacy against a malicious party by relying on the security of a seed shared between two honest parties.

References

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

Sameer Wagh, Divya Gupta, and Nishanth Chandran. Securenn: Efficient and private neural network training. IACR Cryptol. ePrint Arch., 2018:442, 2018.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In 27th USENIX Security Symposium (USENIX Security 18), pages 1651–1669. 2018.

Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pages 35–52, 2018.

Ali Burak Ünal, Mete Akgün, and Nico Pfeifer. A framework with randomized encoding for a fast privacy preserving calculation of non-linear kernels for machine learning applications in precision medicine. In International Conference on Cryptology and Network Security, pages 493–511. Springer, 2019.

Ivan Damgård, Daniel Escudero, Tore Frederiksen, Marcel Keller, Peter Scholl, and Nikolaj Volgushev. New primitives for actively-secure mpc over rings with applications to private machine learning. In 2019 IEEE Symposium on Security and Privacy (SP), pages 1102–1120. IEEE, 2019.

Megha Byali, Harsh Chaudhari, Arpita Patra, and Ajith Suresh. Flash: fast and robust framework for privacy-preserving machine learning. Proceedings on Privacy Enhancing Technologies, 2020(2):459–480, 2020.

Arpita Patra and Ajith Suresh. BLAZE: blazing fast privacy-preserving machine learning. In 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23–26, 2020. The Internet Society, 2020.

Kamalika Chaudhuri and Staal A Vinterbo. A stability-based validation procedure for differentially private machine learning. In Advances in Neural Information Processing Systems, pages 2652–2660, 2013.

Kendrick Boyd, Eric Lantz, and David Page. Differential privacy for classifier evaluation. In Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security, pages 15–23, 2015.

Yan Chen, Ashwin Machanavajjhala, Jerome P Reiter, and Andrés F Barrientos. Differentially private regression diagnostics. In ICDM, pages 81–90, 2016.

Jacob Whitehill. How does knowledge of the auc constrain the set of possible ground-truth labelings? In Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 5425–5432, 2019.
Gregory J Matthews and Ofer Harel. An examination of data confidentiality and disclosure issues related to publication of empirical roc curves. *Academic radiology*, 20(7):889–896, 2013.

Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017. doi: 10.1007/978-3-319-57048-8_6. URL https://doi.org/10.1007/978-3-319-57048-8_6

Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001. doi: 10.1109/SFCS.2001.959888. URL https://doi.org/10.1109/SFCS.2001.959888

Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817, 2016.

Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science, SFCS '86*, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society. ISBN 0-8186-0740-8. doi: 10.1109/SFCS.1986.25. URL https://doi.org/10.1109/SFCS.1986.25

O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC ’87, pages 218–229, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. doi: 10.1145/28395.28420. URL http://doi.acm.org/10.1145/28395.28420

Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Advances in Cryptology - CRYPTO ’91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991*, Proceedings, pages 420–432, 1991. doi: 10.1007/3-540-46766-1\_34. URL https://doi.org/10.1007/3-540-46766-1\_34

Benny Applebaum. Garbled circuits as randomized encodings of functions: a primer. In Yehuda Lindell, editor, *Tutorials on the Foundations of Cryptography*, pages 1–44. Springer International Publishing, 2017. doi: 10.1007/978-3-319-57048-8_1. URL https://doi.org/10.1007/978-3-319-57048-8_1

David P Noren, Byron L Long, Raquel Norel, Kahn Rhiussorakrai, Kenneth Hess, Chenyue Wendy Hu, Alex J Bisberg, Andre Schultz, Erik Engquist, Li Liu, et al. A crowdsourcing approach to developing and assessing prediction algorithms for aml prognosis. *PLoS computational biology*, 12(6):e1004890, 2016.