Prior-Based Label Differential Privacy via Secure Two-Party Computation

Amit Agarwal[*]
University of Illinois
Urbana-Champaign , USA
amita2@illinois.edu

Stanislav Peceny[*]
Georgia Tech, USA
stan.peceny@gatech.edu

Mariana Raykova
Google, USA
marianar@google.com

Phillipp Schoppmann
Google, USA
schoppmann@google.com

Karn Seth
Google, USA
karn@google.com

Abstract

Differential privacy (DP) is a fundamental technique used in machine learning (ML) training for protecting the privacy of sensitive individual user data. In the past few years, a new approach for combining *prior-based* Local Differential Privacy (LDP) mechanisms with a relaxed DP criterion, known as Label DP, has shown great promise in increasing the utility of the final trained model without compromising on the DP privacy budget. In this work, we identify a crucial privacy gap in the current implementations of these prior-based LDP mechanisms, namely the leakage of sensitive priors. We address the challenge of implementing such LDP mechanisms *without* leaking any information about the priors while preserving the efficiency and accuracy of the current insecure implementations. To that end, we design simple and efficient secure two-party computation (2PC) protocols for addressing this challenge, implement them, and perform end-to-end testing on standard datasets such as MNIST, CIFAR-10. Our empirical results indicate that the added security benefit essentially comes almost for "free" in the sense that the gap between the current insecure implementations and our proposed secure version, in terms of run-time overhead and accuracy degradation, is minimal. E.g., for CIFAR-10, with strong DP privacy parameter, the additional runtime due to 2PC is $\approx 3.9\%$ over WAN with 0.4% decrease in accuracy over an insecure (non-2PC) approach.

1 Introduction

Training machine learning (ML) models on user population data is a core capability leveraged in applications that aim to understand user behavior and preferences to be able to make meaningful predictions about them [2, 9, 13, 14, 36, 37]. Of course, the challenge in these scenarios lies in the sensitivity of the input data for training, which belongs to many users. Differential privacy (DP) [29] is a privacy definition, which formalizes the idea that the output of a computation that processes a dataset should not leak information about individual records in that dataset. It has often been used as a privacy measure for the output from ML training on private user data.

[*]Part of this work was done while the author was an intern at Google, USA

There are generally two avenues towards achieving DP for ML training. One of them relies on the notion of local differential privacy (LDP), where each user applies a DP mechanism to its individual contribution and the rest of the training is done as a postprocessing on a differentially private data release. The second approach relies on centralized mechanisms that need as input the original users' data and the computation process itself adds the appropriate randomization to achieve DP output. While the former approach presents a clean privacy argument, LDP mechanisms achieve less utility for the same privacy budget.

In the case of the centralized mechanisms, utility is maximized, but one needs to address the challenges on how to protect the input data while the final DP output is computed. Cryptographic techniques for secure computation (two-party 2PC or multi-party MPC) [10, 38, 67] provide a solution for this question and they have been applied in different contexts with different architectures. In the setting of a single server and many clients, federated learning techniques [9, 13, 14], which leverage the MPC functionality for secure aggregation [14], have been a popular solution. Different approaches rely on several non-colluding servers to execute secure computation protocols for training on the secret-shared data of the clients [5, 42, 58, 62–64]. The latter removes the need for interaction with the users from single-server solutions but adds the non-collusion assumption. Moreover, the efficiency of secure computation for general ML training with even a small number of parties remains a challenge.

Label DP. A recent line of work [18, 33, 36, 37, 54, 61] considers a new notion of differential privacy called *label DP*, where the training features are assumed to be known in the clear by the training party (server), and the goal is to only protect the labels (held by clients). A label DP trained model provides privacy only with respect to the party with the features and it needs to be the only recipient of the output model. This applies to scenarios where a service provider is trying to compute a model over its users' data. It has first party context with its users when the features are generated, but the labels are generated in a different third party context. This is the case in many application scenarios: in user cross-site activities such as online advertising, where features become available at impression time (when a user views an ad) and labels become available during conversion (e.g., when a user buys the advertised product); in recommendation systems where user preferences choices are available to the service providers, but ratings need to be protected; and in medical analyses where patient demographics and general information are available while the test for a particular disease is sensitive. The Private Advertising Technology Working Group (PATWG) at W3C has set forward to solve the question for machine learning training in the setting where the features are known to one party who will receive the model, and thus only the labels need to be protected [57]. We emphasize that in all the aforementioned application scenarios, Label DP is *not* a weaker privacy notion compared to standard DP.

The local differential privacy (LDP) approach can be applied to the setting of label DP by having all users use the randomized response (RR) mechanism [65] to release their labels to the party who knows the features and can then train a label DP model. However, this still suffers from the utility disadvantages of LDP. Ghazi et. al [36, 37] introduced new techniques for randomized response which leverage the prior knowledge about the data, e.g. the distribution of the label values. In particular this solution changes the distribution from which random labels are sampled when the real values are not reported, to be consistent with the prior rather than be uniformly random, i.e., the mechanism is biased by the prior. Ghazi et. al [36] show that the accuracy gain of models trained using prior based randomized response over non-prior based randomized response can be as high as $\approx 8\%$ for moderate DP privacy parameter regimes and tends to increase even further when higher differential privacy is desired.

Need for protecting the privacy of prior. The prior dependent randomized response techniques have been adopted in recent papers [36, 37] to improve the quality of the trained model for classification and regression tasks. These works compute the priors used for the biased randomized response algorithm executed by the clients in different ways. Some of the techniques precompute the priors directly on the input label data with aggregate computation, for example, histograms estimation. Other techniques update the prior throughout the training using both the feature and the label information. Priors that are computed on the sensitive user data, need to provide differential privacy properties.

These existing solutions enforce the training party (holding the features and prior) to leak the sensitive priors to the users/clients in order to enable them to compute their contributions using the biased randomized response algorithm. In most application scenarios, the user population is an open set where any party can pose as a user in the system and participate in the model training protocol. In fact, adversaries can often control a fraction of the users in the system. From this perspective, users should not be trusted to receive any sensitive information about other users as well as the party computing the model. And in the solution using classical local DP mechanisms, the users indeed do not receive any information. While the label differential privacy property of the priors provides a level of protection for the labels of individual users, the approach of giving the priors to the clients effectively increases the information leaked to any arbitrary party compared with pure Local DP setting. This holds even if the features are public because the priors can be an arbitrary function of (DP randomized) labels of *all* the clients. While clients may be comfortable revealing a randomized version of their label to a server, they may be less comfortable if data derived from that label, such as the prior, is shared with all other clients.

Furthermore, while the label DP priors protect individual labels, they do not protect user population level information, which could potentially be sensitive information for the model owner in the case of proprietary models. For example, the prior could be statistics about the users of a certain service which the service owner may not want to reveal to its competitors. There are also settings where the features are not public information but are information that the party building the model knows about the users of its service. In these cases, priors that are computed using the features as well, do not guarantee differential privacy for the individual user features. As a concrete example, the model owner could be an ad tech organization holding the features of a proprietary ads dataset (akin to Criteo dataset [27]) about users and then derives prior-based on this proprietary information. In this case, the priors amount to the model's prediction on that client's features. Since the model is only Label-DP, it may not protect the other features used during training, so revealing this prior to a specific user may leak information about *other* users' features, which are sensitive. Furthermore, for a sensitive model, releasing such prior information to the clients can also leak information about the proprietary dataset and model to competitors, potentially allowing them to gain advantage.

1.1 Our Contributions and Techniques

In this work, we show that revealing the priors to the users is *not* necessary to use the label DP solutions that rely on prior-based randomized response mechanisms. We present a novel solution that leverages secure two-party computation (2PC) between the client and the server that are trying to build the ML model. This enables the server to obtain the client's label value, computed with randomized response with prior [36] mechanism, without the client learning any information about the prior and without the server learning anything more about the real label of the client than the output of prior-based randomized response mechanism. This approach is illustrated in Figure 1.

At the core of our techniques is a novel protocol for joint sampling from a non-uniform distribution where the distribution parameters are known by only one of the two parties (the server holding priors) and should remain secret to the other (the client). To that end, we take a two-fold approach. First, we algorithmically simplify the non-uniform sampling process in prior-based RR mechanisms by representing the underlying distribution as a composition of uniform sampling and a biased bit (Bernoulli) sampling. We do this because it seems non-trivial to construct an *efficient* secure protocol for directly emulating the non-uniform sampling procedure (and we are not aware of any prior work which provides such a protocol). Our representation results in a distribution which is identical to the original distribution while making the sampling procedure more amenable to secure computation. Second, we design modular and efficient sub-protocols to securely sample from these simpler distributions and compose them. All of our protocols make black-box use of Oblivious Transfer (OT) functionality, which is known to be necessary and sufficient for any non-trivial secure computation in the two-party setting [49], and admits a practically efficient realization using IKNP OT extension [45].

One of the main challenges in designing a secure protocol for sampling from these simpler distributions, even after algorithmic simplification, is that the parameter of the distribution is known only to one of the parties. One particularly interesting case is where

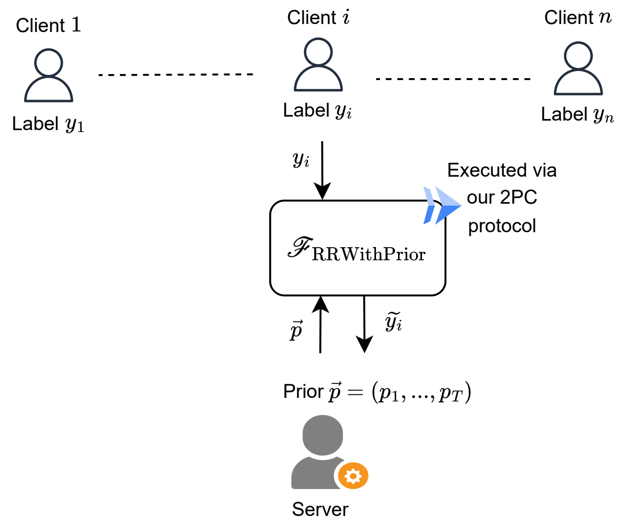
Client 1
Label y_1
Client i
Label y_i
Client n
Label y_n
y_i
Executed via our 2PC protocol
$\mathscr{F}_{\text{RRWithPrior}}$
$\vec{p}$
$\tilde{y}_i$
Prior $\vec{p} = (p_1, ..., p_T)$
Server

Figure 1: Securely randomizing the private label y_i (held by client i) using private prior $\vec{p}$ (held by the server) via an ideal functionality $\mathcal{F}_{\text{RRwithPrior}}$ for which we provide secure 2PC protocol. The functionality outputs the perturbed label $\tilde{y}_i$ to the server. After invoking the functionality between each client and the server, the server obtains a list $\{\tilde{y}_i\}_{i\in[n]}$ of perturbed labels, which can be used for higher-level applications such as model training.

one of the parties (the server) holds a private subset $\mathcal{Y}_{T^*} \subseteq \mathcal{Y}$, where $\mathcal{Y}$ is the public label set and $T^* = |\mathcal{Y}_{T^*}|$. Both parties (client and server) would like to securely sample a random element y from $\mathcal{Y}_{T^*}$ while keeping the sampled output y in secret-shared form and also protecting the input privacy of the set $\mathcal{Y}_{T^*}$ along with its size T^*. We show how to achieve this task efficiently using two sequential calls to OT where the parties exchange their roles in the OT protocol.

We implement our protocol for RR with prior and use it to train standard ML datasets such as MNIST and CIFAR-10. Our protocol does not impose significant communication overhead or latency for the clients. E.g., for training the CIFAR-10 dataset with the privacy budget $\epsilon = 1$ (our most expensive experiment), the runtime cost added due to 2PC over plaintext is ≈ 1.3 minutes, which amounts to $\approx 3.9\%$ overhead. This is a small fraction of the total 33.49min training cost. In total, we communicate 415MB and use 9 rounds. At the same time, our technique does not noticeably degrade the accuracy of the resulting model. Specifically, we observe up to 0.7% difference, which can be reduced with higher fixed-point precision in exchange for a modest increase in the runtime and communication costs (or by using a better model). We also separately benchmark our RR with prior protocol without the training procedure. To compute RR with prior with 10 fractional bits and a label set of size 10, we require 1.49KB of communication and a runtime of 2.2ms on LAN and 320.7ms on WAN.

2 Related Work

Differentially private secure ML training. The notion of differential privacy has been applied to ML training as a way to achieve privacy for the final outputs. The work on DP-SGD [3] presented a method to train differentially private models using stochastic gradient descent. However, this work assumes a trusted party that can execute the DP mechanism.

There are different approaches that avoid the need for a trusted party. E.g., they directly use local DP mechanisms such as randomized response [65] and other variants [4, 34, 36, 37]. Many of them require the clients to have additional knowledge such as priors. Another intermediate approach has been the shuffle DP model, which assumes there is a shuffler that shuffles all LDP contributions, before sending them to the analyzer, who builds models [12, 21, 32].

A different technique is federated learning [9, 13, 14], where the training protocol is an interactive process between the server and the clients. In each iteration, the clients compute a gradient update with respect to the current model they receive. The server then obtains a differentially private aggregate of the gradient updates across clients. Federated learning solutions that avoid placing trust into the server use a secure aggregation protocol [14] that prevents the server from seeing anything but the aggregated gradient (where the clients themselves insert the DP noise).

There are also works that explore running the model training in secure computation across a small number of non-colluding parties [5, 42, 58, 62–64] (some works refer to this as vertical federated learning). A recent paper [68] aims to apply the label DP notion in the setting of distributed MPC training. Such solutions provide less efficiency since they need to execute large functionalities in MPC. Additionally, jointly sampling noise from the correct DP distribution is also a complex functionality and few papers do this in a fully distributed setting.

We note that none of these prior works address the question of secure evaluation of prior-based RR mechanism to enable Label DP functionality. We are the first ones to do so by addressing the unique challenges related to securely and efficiently noising the labels in this setting.

Secure noise generation for DP. Starting from the work of Dwork et. al. [28], many followup works [11, 20, 24, 31, 48, 60, 66] explore the problem of securely sampling noise across two or more parties for various DP relevant distributions. We note that in all of these works, the parameter of the noise distribution is *public* whereas, in our setting, the parameter is *private* to one of the parties. One exception is the work of Choi et. al. [22] which studies the question of noise sampling from private distributions. However, the focus of [22] is to construct protocols where the communication cost is sublinear in the description size of the distribution and their protocols require Fully Homomorphic Encryption (FHE) to achieve this task. In contrast, all of our protocols require Oblivious Transfer (OT) which is computationally much more efficient than FHE.[1]

3 Preliminaries

3.1 Notation

In the context of a classification task, we will use $\mathcal{X}$ and $\mathcal{Y}$ to denote the set of possible features and labels, respectively. We will use T to denote the size of $\mathcal{Y}$ and assume without loss of

[1]We do not focus on the goal of achieving sublinear communication as the description size of our distribution is practically small.

generality that $\mathcal{Y} = \{0, \ldots, T-1\}$. For a given scale (also called fixed-point precision) $f \in \mathbb{N}$, we use $x_{\text{fix}} \in [0, 2^f)$ to denote the fixed-point representation of a fractional real value $x \in [0,1)$ where $x_{\text{fix}} = \left\lfloor x \cdot 2^f \right\rfloor$.

3.2 Secure Two-Party Computation (2PC)

Security model. In 2PC, each party has a private input and they are interested in computing a function on their joint inputs in a way so that both parties only learn the final output without learning any extra information about the other party's input. Our protocol constructions, for different functions of interest, are in this two-party setting and provide *semi-honest security* [38] (also known as passive security). I.e., the parties are assumed to follow the prescribed protocol, but can glean extra information passively from the protocol transcript. We denote the two parties by P_0 and P_1. The security in such a model is formally captured by the standard real/ideal paradigm whereby the view of an adversary in the real-world, which corrupts one of the parties (either P_0 or P_1), can be efficiently simulated in the ideal world where parties interact with a trusted third party (also called the ideal functionality). This third party takes the inputs of all parties and outputs the function evaluated on the joint set of all inputs. A detailed description of the security model is in Appendix A.

Rationale behind the security model. The aforementioned model fits well in our specific Label DP application where the two parties are the server and the client, with a prior and a label as their private inputs, respectively. The function of interest is a randomized response with prior mechanism. Moreover, in practical settings, the server (e.g. Amazon, Google, Meta, etc.) will be obligated by an authority (e.g. government) to follow the protocol instructions, but not collect sensitive client information. Our modeling choice is not merely theoretical; it has been deployed in practice for collecting user-statistics for browser telemetry [1] and measuring advertising conversions [63]. These systems assume one or more semi-honest servers to ensure client privacy.[2]

Offline-online 2PC. A 2PC protocol Π may sometimes be divided into an offline preprocessing phase Π_{offline} (independent of parties' inputs) and an online phase Π_{online} that depends on parties' inputs [15, 46]. The security model still remains the same. This form of offline-online split is mainly aimed towards efficiency as the parties can perform the offline part anytime before the inputs arrive and then store the pre-processed information for future use in the online part. We separately benchmark the offline and online cost of our proposed protocols in addition to presenting the total cost.

Secret sharing. We use $[\![x]\!]^{\mathcal{R}}$ to denote an additive sharing of x in ring $\mathcal{R}$. We drop the superscript $\mathcal{R}$ when it is clear from context. Our default choice of $\mathcal{R}$ in this work is $\mathbb{Z}_T$ where T denotes the number of classification labels. We write $[\![x]\!] = ([\![x]\!]_0, [\![x]\!]_1)$ to denote that P_0 and P_1 get shares $[\![x]\!]_0$ and $[\![x]\!]_1$ respectively, such that $[\![x]\!]_0 + [\![x]\!]_1 = x$ in $\mathcal{R}$. An additive sharing is random if $[\![x]\!]_0$ and $[\![x]\!]_1$ are uniformly distributed in $\mathcal{R}$ subject to $[\![x]\!]_0 + [\![x]\!]_1 = x$. Analogously, we use $\langle b \rangle$ to denote a random XOR-sharing of a bit $b \in \{0,1\}$, consisting of bits $\langle b \rangle_0$ and $\langle b \rangle_1$ such that $\langle b \rangle_0 \oplus \langle b \rangle_1 = b$.

Oblivious transfer functionality. We use $\mathcal{F}_{\mathsf{OT}}^{\binom{N}{1},\mathbb{G}}$ to denote an ideal 1-out-of-N chosen Oblivious Transfer (OT) functionality [17, 56] w.r.t. group $\mathbb{G}$. The sender's input to this functionality are N elements, $s_0, \ldots, s_{N-1}$, where each $s_i \in \mathbb{G}$ whereas the receiver's input is a choice index $c \in [0, N-1]$. The receiver receives the element s_c from the functionality whereas the sender has no output. Our usual choice of $\mathbb{G}$ in this work will be the additive group of $\mathbb{Z}_T$ or $\mathbb{Z}_2$. Using the optimized OT extension in [6], we can generate 1-out-of-2 *Random* OT (ROT) correlations (where the inputs of sender and receiver are uniformly random) using a single round at an amortized cost of λ bits of communication from the sender to the receiver and no communication in the other direction. One can non-interactively convert $\log N$ instances of 1-out-of-2 ROT correlations into a single instance 1-out-of-N ROT correlation using [56]. An instance of 1-out-of-N ROT correlation can then be used to perform 1-out-of-N chosen OT $\mathcal{F}_{\mathsf{OT}}^{\binom{N}{1},\mathbb{G}}$ (where the inputs of sender and receiver are arbitrary) using just 2 rounds - first round from the receiver to sender having $\log N$ bits and second round from the sender to receiver having $N \log |\mathbb{G}|$ bits. [3]

3.3 (Standard) Differential Privacy (DP) and Label Differential Privacy (Label DP)

At a high-level, Differential Privacy (DP) [30] ensures that the output of a randomized algorithm is nearly indistinguishable whether or not any single individual's data is included in the input dataset. Formally, an algorithm $\mathcal{A}$ satisfies ϵ-DP[4] if for all neighboring datasets D and D' (which differ by at most one element) and for any subset of outputs S:

$$\Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} \Pr\left[\mathcal{A}(D') \in S\right]$$

Label Differential Privacy (Label DP) [36, 37] is a relaxation of the standard DP, tailored towards ML tasks, where the privacy guarantee is only required w.r.t dataset labels whereas the dataset features are considered non-sensitive. The formal definition of ϵ-label DP is exactly same as ϵ-DP defined above with a slight modification to the notion of neighbouring datasets: Datasets D and D' are neighbors if differ by at most one *label*. We refer the readers to Appendix B for a more detailed definition.

3.4 Prior-based Randomized Response

3.4.1 *Randomized Response with Prior.*

Ghazi et. al. [36] recently proposed a variant of the Randomized Response (RR) mechanism, termed Randomized Response with Prior ($\mathsf{RRWithPrior}$). In vanilla RR, the private input $y \in \mathcal{Y}$ is randomized by either retaining the same input y with some fixed probability q or replacing

[2]In principle, one can also try to construct protocols in the malicious setting, where we do not assume the corrupt parties follow protocol instructions. Semi-honest model is commonly the first step to obtaining secure protocols in the malicious setting, which is an interesting open problem for future work. Also, in our application, where an authority can enforce parties to comply with the protocol instructions, it might be counter-productive to deploy a maliciously secure protocol as it will add unnecessary runtime and communication overhead.

[3]While it is theoretically possible to reduce the cost of $\mathcal{F}_{\mathsf{OT}}^{\binom{N}{1},}$ to $o(N)$ using techniques based on single server Private Information Retrieval (PIR) [26, 52], the cost benefit only shows up for large values of N. In our experiments, the value of N will be small (on the order of $N \approx 10$) so we do not use heavy-weight PIR based optimization.

[4]The parameter ϵ is typically known as the privacy parameter/budget. A lower value of ϵ implies higher privacy.

with a uniformly random input from $\mathcal{Y}$ with probability $1-q$. In contrast, RRWithPrior, as the name suggests, uses a prior probability distribution $\vec{p} = (p_1, \ldots, p_T)$ to randomize the private input y in a way that maximizes the signal-to-noise ratio while maintaining (ϵ, δ)-DP guarantee w.r.t. y. Note that there is no DP requirement w.r.t. the prior $\vec{p}$.

Algorithm 1: RRWithPrior

```
Parameters: Number of labels $T$, Label set $\mathcal{Y} = \{0, \ldots, T-1\}$,
  privacy parameter $\epsilon \in \mathbb{R}_{\geq 0}$, prior probability vector $\vec{p} = (p_1, \ldots,$
  $p_T)$ where each $p_i \in \mathbb{R}$.
Inputs: Label $y \in \mathcal{Y}$.
Output: A perturbed label $\widetilde{y} \in \mathcal{Y}$.
1  Compute $T^* = \text{argmax}_{t \in [T]} \left( \frac{e^\epsilon}{e^\epsilon + t - 1} \cdot \left( \sum_{i \in \mathcal{Y}_t} p_i \right) \right)$, where $\mathcal{Y}_t$ is
     the set of $t$ labels with maximum $p_i$ values.
2  Let $\mathcal{Y}_{T^*}$ be the set of $T^*$ labels with maximum $p_i$ values.
3  if $y \in \mathcal{Y}_{T^*}$ then
4      Define a random variable $Y_1$ with values in $\mathcal{Y}$ and let $\mathcal{D}_1$ be the
         induced probability distribution s.t.
5      $$\Pr\left[Y_1 = y'\right] = \begin{cases} \frac{e^\epsilon}{e^\epsilon + T^* - 1} & ; y' = y \\ \frac{1}{e^\epsilon + T^* - 1} & ; y' \in \mathcal{Y}_{T^*} \setminus \{y\} \\ 0 & ; y' \in \mathcal{Y} \setminus \mathcal{Y}_{T^*} \end{cases}$$
6      Sample $\widetilde{y} \leftarrow \mathcal{D}_1$
7  else
8      Define a random variable $Y_2$ with values in $\mathcal{Y}$ and let $\mathcal{D}_2$ be the
         induced probability distribution s.t.
       $$\Pr\left[Y_2 = y'\right] = \begin{cases} \frac{1}{T^*} & ; y' \in \mathcal{Y}_{T^*} \\ 0 & ; y' \in \mathcal{Y} \setminus \mathcal{Y}_{T^*} \end{cases}$$
9      Sample $\widetilde{y} \leftarrow \mathcal{D}_2$
10 return $\widetilde{y}$.
```

In Algorithm 1, we describe the randomizer that is proposed in [36]. At a high-level, the algorithm finds a reduced set $\mathcal{Y}_{T^*}$ consisting of T^* labels and then performs vanilla RR on $\mathcal{Y}_{T^*}$ if $y \in \mathcal{Y}_{T^*}$. If $y \notin \mathcal{Y}_{T^*}$, it simply returns a uniformly random label from $\mathcal{Y}_{T^*}$. Ghazi et. al. [36] prove that this approach satisfies ϵ-DP guarantee and also maximizes the objective function capturing the signal-to-noise ratio.

Theorem 3.1 ([36]). *For all $T \in \mathbb{N}, \vec{p} \in \mathbb{R}^T, \epsilon \in \mathbb{R}_{\geq 0}$,* RRWithPrior *(Algorithm 1) is ϵ-DP (Definition B.2).*

Ghazi et. al. [36] also show that in the Label DP setting, training a model using labels perturbed via RRWithPrior algorithm achieves a much better accuracy compared to performing the same training using labels perturbed via vanilla RR mechanism for any fixed privacy budget ϵ.

3.4.2 *Prior computation and model training.*

In Algorithm 2, we describe the general domain-agnostic approach proposed in [36], known as Label Privacy Multi-Stage Training (LP-MST), which makes a black-box use of RRWithPrior as a subroutine. In our experiments, we focus on this general approach for computing the priors and performing model training. The authors show that if RRWithPrior satisfies ϵ-DP guarantee, then LP-MST will satisfy ϵ-label DP guarantee which is captured by Theorem 3.2.[5]

Theorem 3.2 ([36]). *Let $\epsilon \in \mathbb{R}_{\geq 0}$. If* RRWithPrior *(Algorithm 1) is ϵ-DP (Definition B.2), then for all feature sets $\mathcal{X}$, $T \in \mathbb{N}$, $I \in \mathbb{N}$ and for all training algorithms $\mathcal{A}$,* LP-MST *(Algorithm 2) is ϵ-LabelDP (Definition B.4).*

At a high-level, this holds because each private label y_i is queried only once in Line 7 of Algorithm 2 (highlighted in blue) via the ϵ-DP mechanism RRWithPrior.

Algorithm 2: Label Privacy Multi-Stage Training (LP-MST)

```
Parameters: Feature set $\mathcal{X}$, number of labels $T$, label set
  $\mathcal{Y} = \{0, \ldots, T-1\}$, privacy parameter $\epsilon \in \mathbb{R}_{\geq 0}$, number of
  iterations $I$, training algorithm $\mathcal{A}$.
Inputs: Dataset $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ where
  $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ for all $i \in [n]$.
Output: A trained model $M$.
1  Arbitrarily partition $S$ into $S^{(1)}, \ldots, S^{(I)}$
2  Let $M^{(0)}$ be the trivial model that always assigns equal probability
     to each class.
3  for $i = 1$ to $I$ :
4      Let $\widetilde{S}^{(i)} = \phi$
5      for $(x_i, y_i) \in S^{(i)}$ :
6          Let $\vec{p} = (p_1, \ldots, p_T)$ be the probabilities predicted by $M^{(i-1)}$
             on $x_i$
7          Let $\widetilde{y}_i :=$ RRWithPrior$_{T, \mathcal{Y}, \epsilon, \vec{p}}(y_i)$
8          Add $(x_i, \widetilde{y}_i)$ to $\widetilde{S}^{(i)}$
9      Let $M^{(i)}$ be the model resulting from training on $\widetilde{S}^{(1)}, \ldots, \widetilde{S}^{(i)}$
         using $\mathcal{A}$
10 return $M^{(I)}$.
```

4 Securely Computing RR with Prior

Note that classical RR, described in the beginning of Section 3.4, admits a straightforward secure protocol wherein the client locally perturbs its private label y and sends the perturbed label $\widetilde{y}$ to the server[6]. However, the same protocol cannot be used to securely compute prior based RR mechanisms. Specifically, let's focus our attention towards RRWithPrior mechanism described in Section 3.4. In this case, there are *two* private inputs namely the label y (privately held by the client) and prior vector $\vec{p}$ (privately held by the server). Therefore, the protocol needs to operate jointly on the private input held by both parties while ensuring that the client learns nothing about $\vec{p}$ and the server learns nothing about y except $\widetilde{y}$.

Problem formulation. We observe that securely training a neural network in a Label DP setting using LP-MST (Algorithm 2) boils down to securely computing the randomizer RRWithPrior (Algorithm 1). This is because once the private labels are randomly

[5]We remark that while we stick to LP-MST approach in this work for computing the priors in our experiments, the secure protocols that we propose are not limited to it and can work with any prior that the server has independently of how the prior is computed.

[6]This protocol is secure only in the semi-honest setting which is the focus of our work. In the malicious setting, we will need to additionally ensure that the client is adding the DP noise from the right distribution. This is a much harder problem which we leave as a part of future work.

perturbed using RRWithPrior, the perturbed labels can be released in clear to the party holding the features which it can then use for performing training using LP-MST. In this section, we will design an approach to securely execute the RRWithPrior randomizer in a client-server setting. We will denote the server as P_0 and the client as P_1.The setting is as follows: P_0 privately holds a prior vector $\vec{p} \in \mathbb{R}^T$ whereas P_1 privately holds a label $y \in \mathcal{Y}$. They would like to execute Algorithm 1 securely in 2PC so that P_0 learns the output $\widetilde{y}$ (i.e. perturbed label) in the clear. In terms of security, we informally require the following guarantees:

- For a semi-honest P_0, the view is simulatable given only the perturbed label $\widetilde{y}$ and prior vector $\vec{p}$.
- For a semi-honest P_1, the view is simulatable given only the label y.

To capture the privacy requirements more formally, we need to define an ideal functionality which precisely captures the task RRWithPrior (Algorithm 1) that we wish to securely compute. Before doing that, we will simplify the non-uniform sampling step in RRWithPrior (Algorithm 1) so that it is amenable to secure computation without modifying the underlying distribution.

Simplifying the non-uniform sampling. In Algorithm 3, we describe an equivalent and simplified way of performing the non-uniform sampling that happens in RRWithPrior (Steps 3-10 in Algorithm 1). At a high-level, this simplification basically represents the non-uniform distribution induced on $\widetilde{y}$ in Algorithm 1 as a composition of uniform distribution and a biased bit (Bernoulli) distribution. Looking ahead, this simplified sampling procedure will enable a simplified design of the 2PC protocol via modular subprotocols.

Algorithm 3: Simplified sampling in RRWithPrior

```
1 Sample $b_1 \leftarrow \mathsf{Bernoulli}(q)$, where $q = \frac{e^\epsilon - 1}{e^\epsilon + T^* - 1}$.
2 Compute $b_2 := \mathbf{1}\{y \in \mathcal{Y}_{T^*}\}$
3 Sample $z \leftarrow \mathcal{Y}_{T^*}$
4 If $b_1 \wedge b_2 = 1$, set $\widetilde{y} := y$. Else set $\widetilde{y} := z$
5 return $\widetilde{y}$.
```

We now briefly explain why the Steps 1-5 of Algorithm 3 are equivalent to Steps 3-10 of Algorithm 1. Suppose $y \in \mathcal{Y}_{T^*}$. In this case, $b_2 = 1$ and the output $\widetilde{y}$ is y with probability $q + (1-q)\frac{1}{T^*} = \frac{e^\epsilon}{e^\epsilon + T^* - 1}$, an element of $y' \in \mathcal{Y}_{T^*} \setminus \{y\}$ each with probability $(1-q)\frac{1}{T^*} = \frac{1}{e^\epsilon + T^* - 1}$, an element of $y' \in \mathcal{Y} \setminus \mathcal{Y}_{T^*}$ each with probability 0. This distribution matches the distribution $\mathcal{D}_1$ in Algorithm 1. Now let's consider the case where $y \notin \mathcal{Y}_{T^*}$. In this case, $b_2 = 0$ and the output $\widetilde{y}$ is simply z which is a uniformly random element from $\mathcal{Y}_{T^*}$. Hence, this distribution matches the distribution $\mathcal{D}_2$ in Algorithm 1.

Defining the ideal functionality. Now we define an ideal functionality $\mathcal{F}_{\mathsf{RRwithPrior}}$ in Figure 2 which captures RRWithPrior (Algorithm 1) where we replace Steps 3-10 of Algorithm 1 with Steps 1-5 of Algorithm 3 for simplification. The security of a protocol $\Pi_{\mathsf{RRwithPrior}}$ realizing $\mathcal{F}_{\mathsf{RRwithPrior}}$ is captured via the standard simulation based security notion which is described in Appendix A.

Functionality: $\mathcal{F}_{\mathsf{RRwithPrior}}$

Public Parameters: Number of classes T, label set $\mathcal{Y} = \{0, \ldots, T-1\}$, privacy parameter $\epsilon \in \mathbb{R}_{\geq 0}$, fixed point precision $f \in \mathbb{N}$.

(1) Get a prior vector $\vec{p} = \{p_i\}_{i \in [T]}$ as input from P_0 where $p_i \in \mathbb{R}, p_i \in [0,1]$ and $\sum_{i \in [T]} p_i = 1$.
(2) Get a label $y \in \mathcal{Y}$ as input from P_1.
(3) Compute $T^* = \mathsf{argmax}_{t \in [T]} \left(\frac{e^\epsilon}{e^\epsilon + t - 1} \cdot (\sum_{i \in \mathcal{Y}_t} p_i) \right)$, where $\mathcal{Y}_t$ is the set of t labels with maximum p_i values.
(4) Fix $\mathcal{Y}_{T^*}$ to be the set of T^* labels with maximum p_i values.
(5) Set $q_{\mathsf{fix}} = \left\lfloor \frac{e^\epsilon - 1}{e^\epsilon + T^* - 1} \cdot 2^f \right\rfloor$.
(6) Sample $b_1 \leftarrow \mathsf{Bernoulli}(q_{\mathsf{fix}}/2^f)$.
(7) Compute $b_2 := \mathbf{1}\{y \in \mathcal{Y}_{T^*}\}$.
(8) Sample $z \leftarrow \mathcal{Y}_{T^*}$.
(9) If $b_1 \wedge b_2 = 1$, set $\widetilde{y} := y$. Else set $\widetilde{y} := z$.
(10) Send $\widetilde{y}$ to P_0.

Figure 2: Ideal functionality for RRWithPrior

We remark two important differences between $\mathcal{F}_{\mathsf{RRwithPrior}}$ (Figure 2) and RRWithPrior (Algorithm 1). First, $\mathcal{F}_{\mathsf{RRwithPrior}}$ treats the prior vector $\vec{p}$ as an input of party P_0 instead of public parameter. This, as explained earlier, is necessary to capture the privacy requirements of the prior $\vec{p}$ w.r.t. P_0. Second, $\mathcal{F}_{\mathsf{RRwithPrior}}$ introduces an additional parameter f which captures the fixed point precision to be used for representing the bias value q. This is needed because the secure protocol $\Pi_{\mathsf{RRwithPrior}}$ realizing this functionality will be limited to performing computations in a finite fixed-point domain. We note that this restriction to fixed-point domain does not lead to leakage-based attacks [55] and has minimal effect on the ϵ as explained in Section 5.

Protocol construction. We first note that Steps 3-5 in $\mathcal{F}_{\mathsf{RRwithPrior}}$ can be computed locally by P_0 to derive the set $\mathcal{Y}_{T^*}$ and q_{fix} since it only depends on the prior vector $\vec{p}$. The main challenge lies in computing Steps 6 - 10. Note that securely computing these steps requires the P_0's set $\mathcal{Y}_{T^*}$, along with it's size T^*, to be kept private (in addition to the keeping P_1's label y private).

To design $\Pi_{\mathsf{RRwithPrior}}$, a protocol that securely realizes the functionality $\mathcal{F}_{\mathsf{RRwithPrior}}$, we will take a modular approach and design subprotocols for individually computing each step, from Steps 6 - 10, securely. These subprotocols will take either private or secret-shared input and then output the result in a secret-shared form to both parties. Table 1 shows the mapping between the above steps and their corresponding subprotocol names. Since Step 10 has a trivial protocol where both parties do a public reconstruction of $\widetilde{y}$ (from Step 9) towards P_0, we do not explicitly introduce a protocol for it.

4.1 Subprotocols

In this section, we describe the construction of subprotocols from Table 1. To formally prove the security of our subprotocols, we define functionalities $\mathcal{F}_*$ in Appendix D corresponding to each of the subprotocols Π_* where $* \in \{\mathsf{rand}, \mathsf{membership}, \mathsf{sample\text{-}bias\text{-}bit}, \mathsf{mux}\}$. We then show that Π_* securely realizes the functionality $\mathcal{F}_*$ w.r.t a semi-honest adversary corrupting one of the parties.

Steps in $\mathcal{F}_{\mathsf{RRwithPrior}}$	Subprotocols
6	Π_{biasBit}
7	$\Pi_{\mathsf{membership}}$
8	Π_{rand}
9	$\Pi_{\mathsf{mux}}^{\binom{4}{1}}$

Table 1: Mapping between $\mathcal{F}_{\mathsf{RRwithPrior}}$ steps and corresponding sub-protocols implementing them. These sub-protocols would be combined to derive the final protocol $\Pi_{\mathsf{RRwithPrior}}$.

4.1.1 *Secure sampling from a private set.*

In this protocol, we have a public set $\mathcal{Y}_T = \{0, \ldots, T-1\}$ and P_0 has a private set $\mathcal{Y}_{T^*} \subseteq \mathcal{Y}_T$ of size T^* (which is also private). Both parties would like to securely sample (an arithmetic sharing of) a uniformly random element $y \in \mathcal{Y}_{T^*}$.

Protocol. We describe a protocol for realizing this task in Figure 3. This protocol proceeds in two stages. At a high-level, parties first generate a sharing of random value $v \in [0, T^* - 1]$ and then, in the second stage, simply retrieve the sharing of v^{th} element in P_0's private set $\mathcal{Y}_{T^*}$. We now explain each stage in detail. In the first stage, P_1 samples a random mask z and then both parties invoke $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1},\mathbb{Z}_T}$ where P_1 acts as the OT sender and sets the i^{th} sender string to be a fresh random value in $[0, i]$ masked by z. P_0 acts as the OT receiver using $T^* - 1$ as the choice index and ends up retrieving a value $[\![v]\!]_0$. Interpreting P_1's private mask z as a share $[\![v]\!]_1$, it is clear that $([\![v]\!]_0, [\![v]\!]_1)$ form a random sharing of a random value $v \in [0, T^* - 1]$.

In the second stage, both parties again invoke $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1},\mathbb{Z}_T}$ but this time with the sender and receiver roles reversed. The aim of this OT is to retrieve a sharing of the v^{th} element from P_0's private set $\mathcal{Y}_{T^*}$. This can be done by having P_0, acting as the OT sender, set up the OT sender strings in a way so that whenever P_1, acting as the OT receiver, uses $[\![v]\!]_1$ as its choice index, then P_1 retrieves a sharing of the v^{th} element from $\mathcal{Y}_{T^*}$. P_0 can set-up the OT strings in this desired fashion as it knows its own share of v, which is $[\![v]\!]_0$, and can simply iterate over all possible values of $[\![v]\!]_1$ (there are T such possible values which is equal to the number of OT strings).

The protocol as described so far is not actually secure. The reason is that the value v is in the view of P_1 as it was the one who sampled it. As a concrete attack, suppose that all the OT sender strings in the first OT happen to be a masking of 0 (which can happen with some non-negligible probability). Then, P_1 knows that irrespective of what choice index P_0 uses, both will end up with a sharing of $v = 0$ at the end of stage one. This knowledge of v implies that P_1 knows exactly *which* element of P_0's private set $\mathcal{Y}_{T^*}$ will it retrieve (in a shared form) in the second stage. This is a leakage which we do not expect in an ideal setting (see Figure 11, Appendix D.2). To mitigate this leakage, P_0 locally shuffles the set $\mathcal{Y}_{T^*}$ before starting the second stage. At a high-level, this shuffling breaks the correlation between v and the index of element retrieved from $\mathcal{Y}_{T^*}$.

Security and Efficiency. To formally prove security of Π_{rand}, we define an ideal functionality $\mathcal{F}_{\mathsf{rand}}$ (Figure 11,Appendix D.2) and show that Π_{rand} securely realizes $\mathcal{F}_{\mathsf{rand}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1},\mathbb{Z}_T}$ hybrid model (Theorem D.2, Appendix D.2). In terms of efficiency, the protocol simply performs two calls to the OT functionality $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1},\mathbb{Z}_T}$ which incurs 1 round and $2\lambda \log T$ bits of communication in the offline phase. In the online phase, we have 4 rounds (since the two OTs need to be executed sequentially) and $2(T+1)\log T$ bits of communication (across both parties). [7]

Protocol: Π_{rand}

Public input: Set $\mathcal{Y} = \{0, \ldots, T-1\}$ and arithmetic sharing domain $\mathbb{Z}_T$, where $T \in \mathbb{N}$.
Private input: P_0 holds $\mathcal{Y}_{T^*} \subseteq \mathcal{Y}$ where $T^* = |\mathcal{Y}_{T^*}|$.
Output: $[\![r]\!]$ where $r \leftarrow \mathcal{Y}_{T^*}$.

(1) P_1 samples $z \leftarrow \mathbb{Z}_T$ and sets $[\![v]\!]_1 := z$.
(2) P_0 and P_1 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1},\mathbb{Z}_T}$:
- P_1 acts as the OT sender and sets the i^{th} string (0-indexed) as $s_i - z \pmod{T}$, where $s_i \leftarrow [0, i]$.
- P_0 acts as the OT receiver using $T^* - 1$ as the choice index.
- P_0 retrieves $s_{T^*-1} - z$, denoted as $[\![v]\!]_0$.

(3) P_0 computes $\widetilde{\mathcal{Y}}_{T^*} := \pi(\mathcal{Y}_{T^*})$, where π is a random permutation.
(4) P_0 samples $z' \leftarrow \mathbb{Z}_T$.
(5) P_0 and P_1 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1},\mathbb{Z}_T}$:
- P_0 acts as the OT sender and sets the i^{th} OT string (0-indexed) $u_i = d_i - z' \pmod{T}$, where $d_i := ([\![v]\!]_0 + i \pmod{T}))^{\text{th}}$ label in $\widetilde{\mathcal{Y}}_{T^*}$. If $[\![v]\!]_0 + i \pmod{T} \notin \{0, \ldots, T^* - 1\}$, then set $u_i = 0$.
- P_1 acts as the OT receiver with the choice index $[\![v]\!]_1$.
- P_1 retrieves $u_{[\![v]\!]_1}$.

(6) P_0 outputs z'.
(7) P_1 outputs $u_{[\![v]\!]_1}$.

Figure 3: Protocol for secure sampling from a private set

4.1.2 *Secure bit sampling with private bias.*

In this protocol, P_0 holds a private bias value q_{fix} represented in fixed-point format with a public precision f. Both parties would like to securely sample (a boolean sharing of) a biased bit b from the distribution $\mathsf{Bernoulli}(\frac{q_{\mathsf{fix}}}{2^f})$.

Protocol. We describe a protocol for realizing this task in Figure 4. The idea behind this protocol is simple: Both parties invoke $\mathcal{F}_{\mathsf{OT}}^{\binom{2^f}{1},\mathbb{Z}_2}$ where P_0 acts as the OT sender and sets a random subset of $q_{\mathsf{fix}}/2^f$ fraction of the sender strings to be 1 and the rest to be 0. Then, P_1 acts as the OT receiver and simply retrieves a random OT sender string. This will result in P_1 retrieving a bit b from $\mathsf{Bernoulli}(\frac{q_{\mathsf{fix}}}{2^f})$. To have the output bit b in a secret-shared form across both parties, instead of being available in clear to P_1, P_0 can simply mask all the OT sender strings using a random mask bit z and then interpreting z as its share of bit b whereas P_1 interprets the OT retrieved bit as its share of bit b.

[7] As a minor optimization, we can reduce the number of rounds in the online phase from 4 to 3 by parallelizing the second round message of first OT (which is sent by P_1 as it is acting as the OT sender) with the first round message of the second OT (which is again sent by P_1 as it is acting as the OT receiver).

Protocol: Π_{biasBit}

Public input: Fixed point precision $f \in \mathbb{N}$.
Private input: P_0 holds a value $q_{\mathsf{fix}} \in [0, 2^f)$.
Output: $\langle b \rangle$ where $b \leftarrow \mathsf{Bernoulli}(\frac{q_{\mathsf{fix}}}{2^f})$.

(1) P_0 samples $z \leftarrow \{0, 1\}$.
(2) P_0 samples a set $S \subseteq [0, 2^f)$ s.t. $|S| = q_{\mathsf{fix}}$.
(3) P_1 samples $r \leftarrow [0, 2^f)$.
(4) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{OT}}^{\binom{2^f}{1}, \mathbb{Z}_2}$:
  - P_0 acts as the sender. For all $i \in [0, 2^f)$, P_0 sets i^{th} sender bit $s_i = 1 \oplus z$ if $i \in S$ and $0 \oplus z$ otherwise.
  - P_1 acts as the receiver using r as the choice index.
  - P_1 retrieves s_r.
(5) P_0 outputs z.
(6) P_1 outputs s_r.

Figure 4: Protocol for secure bit sampling with private bias.

Security and Efficiency. To formally prove the security of Π_{biasBit}, we define an ideal functionality $\mathcal{F}_{\mathsf{biasBit}}$ (Figure 12 in Appendix D.3) and then show that Π_{biasBit} securely realizes $\mathcal{F}_{\mathsf{biasBit}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{2^f}{1}, \mathbb{Z}_2}$ hybrid model (Theorem D.3 in Appendix D.3). In terms of efficiency, we note that the protocol simply performs a single call to the OT functionality $\mathcal{F}_{\mathsf{OT}}^{\binom{2^f}{1}, \mathbb{Z}_2}$ which incurs 1 round and λf bits of communication in the offline phase and 2 rounds and $2^f + f$ bits of communication (across both parties) in the online phase.

4.1.3 *Secure membership test.*

In this protocol, we have a public set $\mathcal{Y} = \{0, \ldots, T-1\}$ and P_0 has a private set $\mathcal{Y}_{T^*} \subseteq \mathcal{Y}_T$ of size T^*. P_1 has a private element $y \in \mathcal{Y}$ and both parties would like to securely compute (a boolean sharing of) a bit b indicating whether $y \in \mathcal{Y}_{T^*}$.

Protocol. We describe a protocol for realizing this task in Figure 5. The idea behind the protocol is simple: P_0 and P_1 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1}}$, where P_0 acts as a sender and sets the i^{th} OT sender string to be 1 iff $i \in \mathcal{Y}_{T^*}$. Then P_1 simply acts as the OT receiver using y as its choice index. It is easy to see that the output of P_1 will be 1 iff $y \in \mathcal{Y}_{T^*}$. In order to have the result of membership test be in a secret-shared form (insted of revealing it in clear to P_1), P_0 can simply mask its OT sender strings using a random bit z and output z as its share of the result. The OT receiver output of the P_1 together with the output of P_0 (which is z) will then constitute a boolean sharing of the membership test result.

Security and Efficiency. To formally prove the security of $\Pi_{\mathsf{membership}}$, we define an ideal functionality $\mathcal{F}_{\mathsf{membership}}$ (Figure 10 in Appendix D.1) and then show that $\Pi_{\mathsf{membership}}$ securely realizes $\mathcal{F}_{\mathsf{membership}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1}, \mathbb{Z}_2}$ hybrid model (Theorem D.1 in Appendix D.1). In terms of efficiency, the protocol performs a single call to the OT functionality $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1}, \mathbb{Z}_2}$ which incurs 1 round and $\lambda \log T$ bits of communication in the offline phase and 2 rounds and $T + \log T$ bits of communication (across both parties) in the online phase.

Protocol: $\Pi_{\mathsf{membership}}$

Public input: Set $\mathcal{Y} = \{0, \ldots, T-1\}$, where $T \in \mathbb{N}$.
Private inputs: P_0 holds $\mathcal{Y}_{T^*} \subseteq \mathcal{Y}$ where $T^* = |\mathcal{Y}_{T^*}|$ and P_1 holds $y \in \mathcal{Y}$.
Output: $\langle b \rangle$ where $b := \mathbf{1}\{y \in \mathcal{Y}_{T^*}\}$

(1) P_0 samples $z \leftarrow \{0, 1\}$.
(2) P_0 and P_1 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{T}{1}, \mathbb{Z}_2}$:
  - P_0 acts as the OT sender and sets the i^{th} OT bit $s_i = \mathbf{1}\{i \in Y_{T^*}\} \oplus z$
  - P_1 acts as the OT receiver with the choice index y.
  - P_1 retrieves s_y.
(3) P_0 outputs z.
(4) P_1 outputs s_y.

Figure 5: Protocol for secure membership test.

4.1.4 *4-choose-1 secure multiplexer.*

In this protocol, both parties hold an arithmetic sharing of four values, $(x^{0,0}, x^{0,1}, x^{1,0}, x^{1,1})$, over $\mathbb{Z}_T$ and a boolean sharing of two choice bits (c, d). They would like to securely compute an (fresh) arithmetic sharing of $x^{c,d}$.

Protocol. We describe a protocol for realizing this task using $\mathcal{F}_{\mathsf{OT}}^{\binom{4}{1}, \mathbb{Z}_T}$ in Figure 14 in Appendix D.4, which follows from [59].

Security and Efficiency. To formally prove the security of $\Pi_{\mathsf{mux}}^{\binom{4}{1}}$, we define an ideal functionality $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$ (Figure 13 in Appendix D.4) and then state that $\Pi_{\mathsf{mux}}^{\binom{4}{1}}$ securely realizes $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{4}{1}, \mathbb{Z}_T}$ hybrid model (Theorem D.4 in Appendix D.4). In terms of efficiency, the protocol performs two calls to the OT functionality $\mathcal{F}_{\mathsf{OT}}^{\binom{4}{1}, \mathbb{Z}_T}$ which incurs 1 round and 4λ bits of communication in the offline phase and 2 rounds (since the two OTs can be executed in parallel) and $2(4 \log T + 2)$ bits of communication (across both parties) in the online phase.

4.2 Overall Protocol

We will now describe the overall protocol $\Pi_{\mathsf{RRwithPrior}}$ for realizing $\mathcal{F}_{\mathsf{RRwithPrior}}$. To do so, we will assume that parties have access to functionalities $\mathcal{F}_*$ corresponding to the subprotocols Π_* where $* \in \{\mathsf{rand}, \mathsf{membership}, \mathsf{biasBit}, \mathsf{mux}\}$. These functionalities are described in Appendix D.

Protocol. Figure 6 describes our protocol $\Pi_{\mathsf{RRwithPrior}}$ for securely realizing the $\mathcal{F}_{\mathsf{RRwithPrior}}$ functionality introduced earlier. It is a simple combination of the modular sub-protocols as we described earlier in Table 1 followed by a final reconstruction of the perturbed label $\tilde{y}$ in the last step towards P_0. The correctness of the overall protocol follows directly from the correctness of the underlying sub-protocols.

Protocol: $\Pi_{\mathsf{RRwithPrior}}$

Public inputs: Number of classes T, label set $\mathcal{Y} = \{0, \ldots, T-1\}$, privacy parameter $\epsilon \in \mathbb{R}_{\geq 0}$, fixed point precision $f \in \mathbb{N}$, arithmetic sharing domain $\mathbb{Z}_T$.
Private inputs: P_0 holds a prior vector $\vec{p} = \{p_i\}_{i\in[T]}$ where $p_i \in \mathbb{R}, p_i \in [0,1]$ and $\sum_{i\in[T]} p_i = 1$. P_1 holds a label $y \in \mathcal{Y}$.
Output: P_0 outputs a perturbed label $\widetilde{y} \in \mathcal{Y}$.

(1) P_0 computes $T^* = \text{argmax}_{t\in[T]}\left(\frac{e^\epsilon}{e^\epsilon+t-1}\cdot(\sum_{i\in\mathcal{Y}_t} p_i)\right)$, where $\mathcal{Y}_t$ is the set of t labels with maximum p_i values.
(2) P_0 sets $\mathcal{Y}_{T^*}$ to be the set of T^* labels with maximum p_i values.
(3) P_0 computes $q_{\mathsf{fix}} = \left\lfloor \frac{e^\epsilon-1}{e^\epsilon+T^*-1}\cdot 2^f \right\rfloor$
(4) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{biasBit}}$, where P_0 inputs q_{fix}, and they receive shares $[\![b_1]\!]_0$ and $[\![b_1]\!]_1$ respectively.
(5) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{membership}}$, where P_0 inputs $\mathcal{Y}_{T^*}$ and P_1 inputs y, and they receive shares $[\![b_2]\!]_0$ and $[\![b_2]\!]_1$ respectively.
(6) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{rand}}$, where P_0 inputs $\mathcal{Y}_{T^*}$, and they receive shares $[\![z]\!]_0$ and $[\![z]\!]_1$ respectively.
(7) P_0 and P_1 set $[\![y]\!]_0 = 0 \in \mathbb{Z}_T$ and $[\![y]\!]_1 = y \in \mathbb{Z}_T$ respectively, where $[\![y]\!] = ([\![y]\!]_0, [\![y]\!]_1)$ denotes a default sharing of the value y over $\mathbb{Z}_T$ held by P_1.
(8) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$ using selection shares $([\![b_1]\!], [\![b_2]\!])$ and value shares $([\![z]\!], [\![z]\!], [\![z]\!], [\![y]\!])$. They finally receive a sharing of $[\![\widetilde{y}]\!]$, where $\widetilde{y} = y$ if $b_1 \wedge b_2 = 1$, and z otherwise.
(9) P_1 sends $[\![\widetilde{y}]\!]_1$ to P_0. P_0 then reconstructs $\widetilde{y} := [\![\widetilde{y}]\!]_0 + [\![\widetilde{y}]\!]_1 \pmod T$ and outputs it.

Figure 6: Protocol for securely computing RRWithPrior**.**

Security and Efficiency. To formally prove the security of $\Pi_{\mathsf{RRwithPrior}}$, we show (Theorem D.6 in Appendix D.5) that $\Pi_{\mathsf{RRwithPrior}}$ (Figure 6) securely realizes $\mathcal{F}_{\mathsf{RRwithPrior}}$ (Figure 2) in the $\mathcal{F}_{\mathsf{membership}}, \mathcal{F}_{\mathsf{biasBit}}, \mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}, \mathcal{F}_{\mathsf{rand}}$ hybrid model, with respect to passive corruption. The efficiency of $\Pi_{\mathsf{RRwithPrior}}$ will be governed by the efficiency of the underlying sub-protocols namely $\Pi_{\mathsf{membership}}, \Pi_{\mathsf{biasBit}}, \Pi_{\mathsf{mux}}^{\binom{4}{1}}, \Pi_{\mathsf{rand}}$. As each of these sub-protocols is invoked exactly once in $\Pi_{\mathsf{RRwithPrior}}$, the communication cost is simply the sum of the communication cost of each sub-protocol. Therefore, the offline and online communication cost of $\Pi_{\mathsf{RRwithPrior}}$ comes out to be $\lambda(3\log T + f + 4)$ and $\log T(2T + 11) + T + 2^f + f + 4$ bits respectively. In terms of rounds, the offline phase requires a single round as all the sub-protocols just require a single offline round for OT extension which can be performed in parallel.

For online rounds, we note that the sub-protocols for $\mathcal{F}' = \{\mathcal{F}_{\mathsf{biasBit}}, \mathcal{F}_{\mathsf{membership}}, \mathcal{F}_{\mathsf{rand}}\}$ can be invoked in parallel as their inputs and outputs do not have any inter-dependency. However, the sub-protocol for $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$ can only be invoked after the sub-protocols for $\mathcal{F}'$ are completed because the output of $\mathcal{F}'$ is fed as input to $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$. This means that the total online rounds for $\Pi_{\mathsf{RRwithPrior}}$ will be 5 as we need maximum 3 online rounds for the sub-protocols for $\mathcal{F}'$ and 2 online rounds for the sub-protocol for $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$. As an optimization, we note that this can be reduced to 4 rounds instead by parallelizing the third round of Π_{rand} with the first round of $\Pi_{\mathsf{mux}}^{\binom{4}{1}}$. This is possible because the first round of $\Pi_{\mathsf{mux}}^{\binom{4}{1}}$ is only dependent on the output of sub-protocols $\Pi_{\mathsf{biasBit}}, \Pi_{\mathsf{interval}}$, both of which are completed by the second round. As a summary, we present the analytical costs of all our protocols in Table 2.

Theorem 4.1 (Secure RRWithPrior). *Let T be the number of possible labels, f be a fixed point parameter and λ be a security parameter. There exists a protocol $\Pi_{\mathsf{RRwithPrior}}$ (Fig.6) securely realizes $\mathcal{F}_{\mathsf{RRwithPrior}}$ (Fig.2) in the $\mathcal{F}_{\mathsf{OT}}$ hybrid model against semi-honest adversaries using $\approx 2T\log T + 2^f$ bits of online communication and 4 online rounds. Using OT-extension to instantiate $\mathcal{F}_{\mathsf{OT}}$, we have $\approx \lambda(3\log T + f)$ bits of offline communication and a single offline round.*[8]

Proof. We refer the readers to Section 4.2 for efficiency analysis. For security, we take the following approach. First, we define functionalities $\mathcal{F}_*$ in Appendix D corresponding to each of the sub-protocols Π_* where $* \in \{\mathsf{rand}, \mathsf{membership}, \mathsf{sample\text{-}bias\text{-}bit}, \mathsf{mux}\}$ and then show that Π_* securely realizes the functionality $\mathcal{F}_*$ in the $\mathcal{F}_{\mathsf{OT}}$ hybrid model. We then show (Theorem D.6, Appendix D) that $\Pi_{\mathsf{RRwithPrior}}$ (Figure 6) securely realizes $\mathcal{F}_{\mathsf{RRwithPrior}}$ (Figure 2) in the $\mathcal{F}_{\mathsf{membership}}, \mathcal{F}_{\mathsf{biasBit}}, \mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}, \mathcal{F}_{\mathsf{rand}}$ hybrid model. By invoking the UC composability theorem [19], it follows that $\Pi_{\mathsf{RRwithPrior}}$ securely realizes $\mathcal{F}_{\mathsf{RRwithPrior}}$ in the $\mathcal{F}_{\mathsf{OT}}$ hybrid model if the calls to $\mathcal{F}_*$ are replaced by calls to Π_* for $* \in \{\mathsf{rand}, \mathsf{membership}, \mathsf{sample\text{-}bias\text{-}bit}, \mathsf{mux}\}$.

Note that RRWithPrior mechanism is aimed towards adding Label DP guarantee for model training on classification tasks. Ghazi et. al.[37] recently proposed a variant called Randomized Response on Bins (RROnBins) which handles regression tasks. While the focus of this work is specifically on RRWithPrior, our techniques can also be extended to securely compute RROnBins. Due to space limitation, we refer the readers to Appendix C for details.

5 Experimental Evaluation

Implementation Details. We implement our system in C++ and Python 3. Our system implements the proposed secure protocol $\Pi_{\mathsf{RRwithPrior}}$ (Figure 6) and applies it to train neural networks with LabelDP using LP-MST (Algorithm 2). C++ coordinates the whole computation and executes the 2PC protocols. At each LP-MST iteration, the C++ engine makes two invocations to Python scripts that use the Keras [23] interface for neural networks with TensorFlow [16] backend: (1) forward propagation to compute the priors (Line 6 in Algorithm 2), and (2) model training on the randomized labels (Line 9 in Algorithm 2). We embed Python in C++ using C++'s default application programmer's interface to Python (see the Python.h header). This architecture enables us to use Python's convenient libraries for deep learning and linear algebra, including Keras and TensorFlow, and support any (Keras-implemented) model. We note some interesting aspects of our implementation:

- We store the neural network architecture, the model weights, and the training configuration (e.g. loss function, optimizer, etc.) in HDF5 file format. Keras' API provides functions to load from and

[8]This excludes the cost of "base OTs" as it is amortized away over large number of OT instances.

Protocol/Costs	Online communication (bits)	Online rounds	Offline communication (bits)
$\Pi_{\text{membership}}$	$T + \log T$ 14	2	$\lambda \log T$ 512
Π_{rand}	$2(T+1)\log T$ 88	3	$2\lambda \log T$ 1024
Π_{biasBit}	$2^f + f$ 1034	2	λf 1280
$\Pi_{\text{mux}}^{\binom{4}{1}}$	$8\log T + 4$ 36	2	4λ 512
$\Pi_{\text{RRwithPrior}}$	$\log T(2T+11) + T + 2^f + f + 4$ 1172	4	$\lambda(3\log T + f + 4)$ 3328

Table 2: Analytical costs of $\Pi_{\text{RRwithPrior}}$ sub-protocols. The blue highlighted numbers indicate concrete costs for $T = 10$, $\lambda = 128$ and $f = 10$. All protocols require a single offline round (due to OT extension). The costs presented here do not include the cost of "base OTs" as it is amortized away over large number of OT instances. The logarithm function $\log$ is w.r.t base 2.

save to this format. Note that this approach is highly modular as it allows us to easily switch to a completely different neural network without having to modify code.

- In our codebase, we include an efficient implementation of IKNP OT extension [45], including the state-of-the-art optimizations [6, 41]. We precompute a large number of random 1-out-of-2 OTs in the offline phase, and convert them into 1-out-of-N random OTs in a straightforward way via $\log N$ 1-out-of-2 random OTs. In the online phase, we then construct 1-out-of-N chosen OTs from the precomputed 1-out-of-N random OTs with Beaver's classic trick [8].
- Our system includes server/client code with network communication implemented using gRPC's asynchronous APIs.
- Our implementation has end-to-end tests written in the GoogleTest library [40].
- Finally, we use the Bazel build system [39].

Datasets. We run our experiments on 2 datasets. MNIST [25] consists of 28×28 image inputs (i.e. 784 features) and 10 classes. The training set contains $60,000$ examples; the testing set contains $10,000$. CIFAR-10 [50] consists of 32×32 image inputs with 3 channels (i.e. 3072 features) and 10 classes. The training set contains $50,000$ examples; the testing set contains $10,000$.

Network Architecture and Training Configuration. For MNIST, we use a simple convolutional neural network (CNN) that starts with two convolutional layers, each followed by max-pooling to reduce the spatial dimensions of the feature maps. After flattening the output, we use a dropout layer to reduce overfitting, and apply a dense softmax layer to predict one of the 10 digit classes. We compile the model using categorical cross-entropy loss and the Adam optimizer. We train the model using 90% of the dataset, with 10% reserved for validation, over 15 epochs and a batch size of 128. For CIFAR-10, we implement a ResNet-18 V2 architecture with data augmentation and a custom learning rate scheduler. We start with convolutional layers, followed by ResNet blocks that downsample the spatial dimensions while increasing filter counts, and we conclude with global average pooling and a softmax output for classification. To improve generalization, we apply data augmentation techniques such as random horizontal flips and width/height shifts. We train the model using the SGD optimizer with a momentum of 0.9 and an initial learning rate of 0.4, which decays over time through a custom learning rate schedule. The learning rate ramps up during the first 30 epochs before decaying in later stages. We train the model for 35 epochs with a batch size of 512. For all experiments, we use 2 LP-MST iterations and split the dataset equally between these iterations.

Experimental setup. We run our MNIST experiments on two Ubuntu 20.04 LTS compute-optimized c2-standard-8 Google Cloud instances with 32 GB RAM and Intel Xeon CPU at 3.1 GHz clock rate. For CIFAR-10, the machine that runs the deep learning training and prior computation is replaced with an Ubuntu 20.04 LTS g2-standard-32 Google Cloud instance with a single NVIDIA L4 GPU, 128GB RAM, and Intel Xeon CPU at 2.2GHz clock rate. In the LAN setting, we deploy both instances in the us-central1 region, with 0.12ms mean network latency and $\approx$ 1.9GB/s bandwidth. In the WAN setting, we deploy one instance in us-central1 and the other in us-west1. The mean network latency is 31.30ms and the bandwidth $\approx$ 93MB/s. The client and the server execute their computation simultaneously in each round. They proceed to the next round once both parties have completed the computation and transferred the messages to one another. As LP-MST is a randomized algorithm (due to RRWithPrior and Keras model training), we take mean averages over 10 runs for more consistent results.

Goal of our evaluation. [36] already demonstrates that RRWithPrior provides more accurate results than vanilla RR (and other approaches) while maintaining the same level of privacy. We do not repeat their experiments. The goal of our evaluation is to show that RRWithPrior, when replaced with our proposed secure version $\Pi_{\text{RRwithPrior}}$, *does not introduce significant runtime overhead or accuracy loss* in the context of LP-MST training. In other words, given a fixed model architecture, using $\Pi_{\text{RRwithPrior}}$ to randomize the labels (rather than randomizing them in cleartext using RRWithPrior) does not significantly reduce the overall training accuracy or increase the running time of LP-MST. Hence, we do not try to find the most performant network architecture and training parameters (which is purely an ML task). We also note that the running time and communication cost of our $\Pi_{\text{RRwithPrior}}$ is independent of the model as $\Pi_{\text{RRwithPrior}}$ is only used to randomize each label in the dataset exactly once.

LP-MST accuracy experiments. We now show that $\Pi_{\text{RRwithPrior}}$ does not significantly degrade the accuracy of LP-MST w.r.t. plaintext RRWithPrior. We can see from Table 3 that for all experiments the accuracy differs by at most 1%. Note that a larger number of fractional bits f can decrease this difference at the expense of increasing the runtime and communication cost (as we will discuss next).

	MNIST (15 Keras epochs)			**CIFAR-10** (35 Keras epochs)		
	$\epsilon = 1$					
Fixed point precision f	12			13		
Accuracy	93.8% (93.7%)			52.6 % (53.0%)		
	Offline	**Online**	**Total**	**Offline**	**Online**	**Total**
Communication	25.69 MB	237.00 MB	262.69 MB	22.18 MB	392.82 MB	415.00 MB
LAN: 2PC Only	19.12 sec	15.32 sec	34.44 sec	32.63 sec	30.06 sec	62.69 sec
LAN: Full Training	—	—	3.67 min	—	—	33.27 min
WAN: 2PC Only	19.70 sec	19.24 sec	38.94 sec	33.42 sec	42.36 sec	75.78 sec
WAN: Full Training	—	—	3.75 min	—	—	33.49 min
	$\epsilon = 3$					
Fixed point precision f	10			12		
Accuracy	98.5% (98.6%)			88.1% (88.7%)		
	Offline	**Online**	**Total**	**Offline**	**Online**	**Total**
Communication	23.86 MB	61.22 MB	85.08 MB	21.42 MB	197.50 MB	218.92 MB
LAN: 2PC Only	4.15 sec	4.76 sec	8.91 sec	16.09 sec	14.69 sec	30.78 sec
LAN: Full Training	—	—	3.25 min	—	—	32.74 min
WAN: 2PC Only	4.82 sec	6.64 sec	11.46 sec	16.82 sec	19.24 sec	36.06 sec
WAN: Full Training	—	—	3.29 min	—	—	32.83 min
	$\epsilon = 8$					
Fixed point precision f	8			10		
Accuracy	99.1% (99.1%)			92.8% (93.5%)		
	Offline	**Online**	**Total**	**Offline**	**Online**	**Total**
Communication	22.03 MB	17.26 MB	39.29 MB	19.89 MB	51.02 MB	70.91 MB
LAN: 2PC Only	1.18 sec	2.24 sec	3.42 sec	3.67 sec	4.62 sec	8.29 sec
LAN: Full Training	—	—	3.16 min	—	—	32.37 min
WAN: 2PC Only	1.77 sec	3.72 sec	5.49 sec	4.49 sec	7.02 sec	11.51 sec
WAN: Full Training	—	—	3.19 min	—	—	32.42 min

Table 3: We benchmark LP-MST for different ϵ (DP privacy budget) and f (fixed point precision). We fix the number of LP-MST iterations to 2. We compare test dataset accuracy of LP-MST with $\Pi_{\text{RRwithPrior}}$ against LP-MST with plaintext RRWithPrior (in parentheses). We present runtime (LAN and WAN) and communication for the offline phase, online phase, and total (offline + online). In all executions, offline phase uses one round of interaction, while the online phase requires 8. All wall-clock times represent the *maximum* of the client and server; all communication costs represent the *sum* of the client and server. "2PC Only" represents the running time overhead of only $\Pi_{\text{RRwithPrior}}$; "Full Training" represents the total runtime of LP-MST including $\Pi_{\text{RRwithPrior}}$.

LP-MST runtime and communication experiments. As shown in the '2PC Only' rows of Table 2, the cost of $\Pi_{\text{RRwithPrior}}$ during full training in our most computationally intensive CIFAR-10 experiment is less than 1.3 minutes, whereas the total training time exceeds 30 minutes. Our 2PC requires 1 round of interaction to preprocess OTs in the offline phase and 4 rounds per online iteration, which is consistent with our analytical estimates in Table 2.

Scalability of our experiments. Note that the ML training happens outside 2PC on the server side on the noisy labels received from each client. The only part executed inside 2PC is the prior-based RR mechanism, which securely computes and releases the noisy labels to the server. Beyond this one-shot 2PC interaction between each client and the server, the remaining computation on the server side is identical to the traditional ML training on plaintext (but DP perturbed) labels. This leads to the following conclusions: i) Each client's communication/computation cost is independent of the total number of clients in the system. However, the server's communication/computation cost across all clients scales linearly with the number of clients as expected. ii) Since the number of clients is identical to the training set size (as each client is holding a single label), the cost dependency on the training set size is identical to the cost dependency on the number of clients. iii) The size/complexity of the ML model does not affect the 2PC communication/computation cost of the server. It only affects the running time and memory consumption of the "non-2PC" part, which is

Fixed-point precision (f)	8	10	15	20
Comm Offline	384B	416B	496B	576B
Comm Online	349B	1.09KB	32.09KB	1MB
Comm Total	0.72KB	1.49KB	32.57KB	1MB
Rounds Offline	1	1	1	1
Rounds Online	4	4	4	4
LAN Online (ms)	2.11	2.21	4.54	88.80
WAN Online (ms)	315.08	320.66	365.97	661.62

Table 4: This table shows the cost of our system executing $\Pi_{\mathsf{RRwithPrior}}$ for a single input with 8, 10, 15, & 20 fractional bits, assuming 10 possible labels, between a client and a server. The communication is across both parties (client and server) combined.

ML training, in the same way that the model size would affect the time/memory cost of vanilla ML training.

Microbenchmarking $\Pi_{\mathsf{RRwithPrior}}$. We now show in Table 4 the runtime and the communication cost (i.e. both client and server combined) of executing $\Pi_{\mathsf{RRwithPrior}}$ on a single label between a server and a client, for different values of the fixed-point precision f. We assume the number of labels is 10. In our experiments, we observe that LAN and WAN *offline* runtimes are extremely small. For example, with $f = 8$, the LAN and WAN runtimes are 0.02ms and 0.07ms, respectively. As another data point, with $f = 10$, the LAN and WAN runtimes are 0.03ms and 0.08ms, respectively. Therefore, for brevity, we do not include the offline runtimes in Table 4. Table 4 shows that the communication and online runtime remain less than a MB and a second, respectively, even for f as high as 20. Thus, our protocol is suitable for execution on computationally weak devices such as smart phones.

Impact of fixed-point arithmetic on DP. Recent works have shown that implementations of differential privacy mechanisms require care to prevent against leakage due to imprecision of concrete numerical implementations. Mironov [55] was the first to show concrete attacks for mechanisms such as the Laplace distribution, relying on concrete floating point implementations. Many other works [7, 35, 43, 44, 47] have explored this space presenting both attacks and mitigations.

We point out that our approaches are exempt from the attacks on floating point implementations described above since they make use of Bernoulli sampling, namely flipping a coin with probability p in Π_{biasBit}, or sampling at random from T classes in Π_{rand}. However, even Bernoulli sampling can be subject to bias in a fixed-point implementation. For Π_{rand} we avoid this bias by having each party sample a large random string (more than 40 bits longer than $\log(T)$, and taking it modulo T. The result is 2^{-40}-statistically-close to a random sample modulo T.

In Π_{biasBit} we incur a small bias by our use of fixed-point precision of 20 in the representation of the probability p. This bias increases the effective epsilon by a small amount (less than 10^{-3} for the ϵ's considered in the paper), because the probability in the numerator in the DP expression may have increased by up to 2^{-20}, and that in the denominator reduced by 2^{-20}. We consider this small increase negligible. We note that it becomes more significant for $\epsilon > 12$, in which case we would need more than 20 bits.

6 Acknowledgments

We thank Lawrence Roy for helpful discussions.

References

[1] 2023. Privacy-preserving metrics for Firefox. https://divviup.org/blog/divvi-up-in-firefox/.

[2] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[3] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*.

[4] Jayadev Acharya, Kallista Bonawitz, Peter Kairouz, Daniel Ramage, and Ziteng Sun. 2020. Context Aware Local Differential Privacy. In *Proceedings of the 37th International Conference on Machine Learning*.

[5] Nitin Agrawal, Ali Shahin Shamsabadi, Matt J. Kusner, and Adrià Gascón. [n. d.]. QUOTIENT: Two-Party Secure Neural Network Training and Prediction. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.).

[6] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 535–548.

[7] Victor Balcer and Salil Vadhan. 2019. Differential Privacy on Finite Computers. *Journal of Privacy and Confidentiality* 9, 2 (2019).

[8] Donald Beaver. 1995. Precomputing Oblivious Transfer. In *CRYPTO'95 (LNCS, Vol. 963)*, Don Coppersmith (Ed.). Springer, Heidelberg, 97–109. doi:10.1007/3-540-44750-4_8

[9] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. In *CCS*. ACM, 1253–1269.

[10] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *20th ACM STOC*. ACM Press, 1–10. doi:10.1145/62212.62213

[11] Ari Biswas and Graham Cormode. 2023. Interactive proofs for differentially private counting. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 1919–1933.

[12] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. 2017. Prochlo: Strong Privacy for Analytics in the Crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*.

[13] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *MLSys*. mlsys.org.

[14] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *CCS*. ACM, 1175–1191.

[15] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography: 17th International Conference, TCC 2019, Nuremberg, Germany, December 1–5, 2019, Proceedings, Part I 17*. Springer, 341–371.

[16] Google Brain. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/.

[17] Gilles Brassard, Claude Crépeau, and Jean-Marc Robert. 1986. All-or-nothing disclosure of secrets. In *Conference on the Theory and Application of Cryptographic Techniques*. Springer, 234–238.

[18] Robert Istvan Busa-Fekete, Andres Munoz Medina, Umar Syed, and Sergei Vassilvitskii. 2023. Label differential privacy and private training data release. In *International Conference on Machine Learning*. PMLR, 3233–3251.

[19] Ran Canetti. 2001. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE, 136–145.

[20] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. 2019. Securely sampling biased coins with applications to differential privacy. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 603–614.

[21] Albert Cheu, Adam D. Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. 2019. Distributed Differential Privacy via Shuffling. In *EUROCRYPT 2019, Part I (LNCS, Vol. 11476)*, Yuval Ishai and Vincent Rijmen (Eds.). Springer, Heidelberg, 375–403. doi:10.1007/978-3-030-17653-2_13

[22] Seung Geol Choi, Dana Dachman-Soled, S Dov Gordon, Linsheng Liu, and Arkady Yerukhimovich. 2022. Secure sampling with sublinear communication. In *Theory of Cryptography Conference*. Springer, 348–377.

[23] François Chollet et al. 2015. Keras. https://keras.io.

[24] Sankha Das, Sayak Ray Chowdhury, Nishanth Chandran, Divya Gupta, Satya Lokam, and Rahul Sharma. 2024. Communication Efficient Secure and Private Multi-Party Deep Learning. *Cryptology ePrint Archive* (2024).

[25] Li Deng. 2012. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[26] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. 2000. Single database private information retrieval implies oblivious transfer. In *Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19*. Springer, 122–138.

[27] Diemert Eustache, Betlei Artem, Christophe Renaudin, and Amini Massih-Reza. 2018. A Large Scale Benchmark for Uplift Modeling. In *Proceedings of the AdKDD and TargetAd Workshop, KDD, London,United Kingdom, August, 20, 2018*. ACM.

[28] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28-June 1, 2006. Proceedings 25*. Springer, 486–503.

[29] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC 2006 (LNCS, Vol. 3876)*, Shai Halevi and Tal Rabin (Eds.). Springer, Heidelberg, 265–284. doi:10.1007/11681878_14

[30] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography: Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006. Proceedings 3*. Springer, 265–284.

[31] Reo Eriguchi, Atsunori Ichikawa, Noboru Kunihiro, and Koji Nuida. 2022. Efficient noise generation protocols for differentially private multiparty computation. *IEEE Transactions on Dependable and Secure Computing* 20, 6 (2022), 4486–4501.

[32] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by Shuffling: From Local to Central Differential Privacy via Anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '19)*.

[33] Hossein Esfandiari, Vahab Mirrokni, Umar Syed, and Sergei Vassilvitskii. 2022. Label differential privacy via clustering. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 7055–7075.

[34] Mani Malek Esmaeili, Ilya Mironov, Karthik Prasad, Igor Shilov, and Florian Tramèr. 2021. Antipodes of Label Differential Privacy: PATE and ALIBI. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.).

[35] Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. 2016. Preserving differential privacy under finite-precision semantics. *Theoretical Computer Science* 655 (2016), 92–108.

[36] Badih Ghazi, Noah Golowich, Ravi Kumar, Pasin Manurangsi, and Chiyuan Zhang. 2021. Deep learning with label differential privacy. *Advances in neural information processing systems* 34 (2021), 27131–27145.

[37] Badih Ghazi, Pritish Kamath, Ravi Kumar, Ethan Leeman, Pasin Manurangsi, Avinash Varadarajan, and Chiyuan Zhang. 2022. Regression with Label Differential Privacy. *arXiv preprint arXiv:2212.06074* (2022).

[38] Oded Goldreich. 2009. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press.

[39] Google. 2022. Bazel. https://bazel.build/.

[40] Google. 2023. GoogleTest – Google Testing and Mocking Framework. https://github.com/google/googletest.

[41] Chun Guo, Jonathan Katz, Xiao Wang, and Yu Yu. 2020. Efficient and secure multiparty computation from fixed-key block ciphers. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 825–841.

[42] Gauri Gupta, Krithika Ramesh, Anwesh Bhattacharya, Divya Gupta, Rahul Sharma, Nishanth Chandran, and Rijurekha Sen. 2023. End-to-end Privacy Preserving Training and Inference for Air Pollution Forecasting with Data from Rival Fleets. In *Privacy Enhancing technologies Symposium (PETS) 2023*.

[43] Naoise Holohan and Stefano Braghin. 2021. Secure random sampling in differential privacy. In *Computer Security–ESORICS 2021: 26th European Symposium on Research in Computer Security, Darmstadt, Germany, October 4–8, 2021, Proceedings, Part II 26*. Springer, 523–542.

[44] Christina Ilvento. 2020. Implementing the exponential mechanism with base-2 differential privacy. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 717–742.

[45] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference*. Springer, 145–161.

[46] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. 2013. On the power of correlated randomness in secure computation. In *Theory of Cryptography: 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*. Springer, 600–620.

[47] Jiankai Jin, Eleanor McMurtry, Benjamin IP Rubinstein, and Olga Ohrimenko. 2022. Are we there yet? timing and floating-point attacks on differential privacy systems. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 473–488.

[48] Hannah Keller, Helen Möllering, Thomas Schneider, Oleksandr Tkachenko, and Liang Zhao. 2024. Secure Noise Sampling for DP in MPC with Finite Precision. In *Proceedings of the 19th International Conference on Availability, Reliability and Security*. 1–12.

[49] Joe Kilian. 1988. Founding crytpography on oblivious transfer. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*. 20–31.

[50] Alex Krizhevsky. 2009. CIFAR-10 (Canadian Institute for Advanced Research). (2009). http://www.cs.toronto.edu/~kriz/cifar.html

[51] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. 2006. Information-theoretically secure protocols and security under composition. In *Proceedings of the thirty-eighth annual ACM Symposium on Theory of Computing*. 109–118.

[52] Eyal Kushilevitz and Rafail Ostrovsky. 1997. Replication is not needed: Single database, computationally-private information retrieval. In *Proceedings 38th annual symposium on foundations of computer science*. IEEE, 364–373.

[53] Yehuda Lindell. 2017. How to simulate it–a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* (2017), 277–346.

[54] Mani Malek Esmaeili, Ilya Mironov, Karthik Prasad, Igor Shilov, and Florian Tramer. 2021. Antipodes of label differential privacy: Pate and alibi. *Advances in Neural Information Processing Systems* 34 (2021), 6934–6945.

[55] Ilya Mironov. 2012. On significance of the least significant bits for differential privacy. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 650–661.

[56] Moni Naor and Benny Pinkas. 1999. Oblivious transfer and polynomial evaluation. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*. 245–254.

[57] PATCG. 2023. Private Conversion Optimisation. https://github.com/patcg/meetings/issues/117.

[58] Deevashwer Rathee, Anwesh Bhattacharya, Divya Gupta, Rahul Sharma, and Dawn Song. 2023. Secure Floating-Point Training. In *Usenix Security Symposium 2023*.

[59] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 325–342. doi:10.1145/3372297.3417274

[60] César Sabater, Florian Hahn, Andreas Peter, and Jan Ramon. 2023. Private sampling with identifiable cheaters. *Proceedings on Privacy Enhancing Technologies* 2023, 2 (2023).

[61] XU Shirong, Chendi Wang, Will Wei Sun, and Guang Cheng. 2023. Binary classification under local label differential privacy using randomized response mechanisms. *Transactions on Machine Learning Research* (2023).

[62] Kunal Talwar, Shan Wang, Audra McMillan, Vojta Jina, Vitaly Feldman, Bailey Basile, Aine Cahill, Yi Sheng Chan, Mike Chatzidakis, Junye Chen, Oliver Chick, Mona Chitnis, Suman Ganta, Yusuf Goren, Filip Granqvist, Kristine Guo, Frederic Jacobs, Omid Javidbakht, Albert Liu, Richard Low, Dan Mascenik, Steve Myers, David Park, Wonhee Park, Gianni Parsa, Tommy Pauly, Christian Priebe, Rehan Rishi, Guy Rothblum, Michael Scaria, Linmao Song, Congzheng Song, Karl Tarbe, Sebastian Vogt, Luke Winstrom, and Shundong Zhou. 2023. Samplable Anonymous Aggregation for Private Federated Data Analysis. arXiv:2307.15017 [cs.CR]

[63] Erik Taubeneck, Ben Savage, and Martin Thomson. 2022. Interoperable Private Attribution (IPA). https://docs.google.com/document/d/1KpdSKD8-Rn0bWPTu4UtK54ks0yv2j22pA5SrAD9av4s.

[64] Sameer Wagh, Divya Gupta, and Nishanth Chandran. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. *Proc. Priv. Enhancing Technol.* 2019, 3 (2019), 26–49.

[65] Stanley L Warner. 1965. Randomized response: A survey technique for eliminating evasive answer bias. *J. Amer. Statist. Assoc.* 60, 309 (1965), 63–69.

[66] Genqiang Wu, Yeping He, Jingzheng Wu, and Xianyao Xia. 2016. Inherit differential privacy in distributed setting: Multiparty randomized function computation. In *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 921–928.

[67] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th FOCS*. IEEE Computer Society Press, 162–167. doi:10.1109/SFCS.1986.25

[68] Sen Yuan, Milan Shen, Ilya Mironov, and Anderson Nascimento. 2021. Label private deep learning training based on secure multiparty computation and differential privacy. In *NeurIPS 2021 Workshop Privacy in Machine Learning*.

A Semi-Honest Secure Two-Party Computation

The following description of semi-honest two-party computation is standard in the literature and is taken from [53].

Semi-honest adversary. The model that we consider in this work is that of two-party computation in the presence of *static semi-honest* adversaries. Such an adversary controls one of the parties (statically, and so at the onset of the computation) and follows the protocol specification exactly. However, it may try to learn more information than allowed by looking at the transcript of messages that it received and its internal state. A protocol that is secure in the presence of semi-honest adversaries guarantees that there is no inadvertent leakage of information. Semi-honest secure protocols are often designed as the first step towards achieving the stronger notions of malicious security.

Two-party computation (2PC). A two-party protocol problem is cast by specifying a possibly random process that maps pairs of inputs to pairs of outputs (one for each party). We refer to such a process as a *functionality* and denote it $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs $x, y \in \{0,1\}^*$, the output-pair is a random variable $(f_1(x,y), f_2(x,y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x,y)$ and the second party (with input y) wishes to obtain $f_2(x,y)$.

Privacy by simulation. As expected, we wish to formalize the idea that a protocol is secure if whatever can be computed by a party participating in the protocol can be computed based on its input and output only. This is formalized according to the simulation paradigm by requiring the existence of a simulator who generates the view of a party in the execution. However, since the parties here have input and output, the simulator must be given a party's input and output in order to generate the view. Thus, security here is formalized by saying that a party's view in a protocol execution be simulatable given its input and output. This formulation implies that the parties learn nothing from the protocol execution beyond what they can derive from their input and prescribed output.

One important point to note is that since the parties are semi-honest, it is guaranteed that they use the actual inputs written on their input tapes. This is important since it means that the output is well-defined, and not dependent on the adversary. Specifically, for inputs x, y, the output is defined to be $f(x,y)$, and so the simulator can be given this value.

Definition of security. We begin with the following notation:

- Let $f = (f_1, f_2)$ be a probabilistic polynomial-time functionality and let π be a two-party protocol for computing f. (Throughout, whenever we consider a functionality, we always assume that it is polynomially-time computable.)
- The view of the i^{th} party ($i \in \{1,2\}$) during an execution of π on (x,y) and security parameter λ is denoted by $\text{view}_i^{\pi}(x,y,\lambda)$ and equals $(w, r_i; m_1^i, \ldots, m_t^i)$, where $w \in \{x, y\}$ (its input depending on the value of i), r_i equals the contents of the i^{th} party's internal random tape, and m_j^i represents the j^{th} message that it received.
- The output of the i^{th} party during an execution of π on (x,y) and security parameter λ is denoted by $\text{output}_i^{\pi}(x,y,\lambda)$ and can be computed from its own view of the execution. We denote the joint output of both parties by $\text{output}^{\pi}(x,y,\lambda) = (\text{output}_1^{\pi}(x,y,\lambda), \text{output}_2^{\pi}(x,y,\lambda))$.

Definition A.1 (Standalone security). Let $f = (f_1, f_2)$ be a functionality. We say that a protocol π securely computes f in the presence of static semi-honest adversaries in the standalone model if there exist probabilistic polynomial-time algorithms $\mathcal{S}_1$ and $\mathcal{S}_2$ such that

$$\left\{(\mathcal{S}_1(1^{\lambda}, x, f_1(x,y)), f(x,y))\right\}_{x,y,\lambda} \overset{c}{\equiv} \left\{(\text{view}_1^{\pi}(x,y,\lambda), \text{output}^{\pi}(x,y,\lambda))\right\}_{x,y,\lambda}$$

, and

$$\left\{(\mathcal{S}_2(1^{\lambda}, x, f_2(x,y)), f(x,y))\right\}_{x,y,\lambda} \overset{c}{\equiv} \left\{(\text{view}_2^{\pi}(x,y,\lambda), \text{output}^{\pi}(x,y,\lambda))\right\}_{x,y,\lambda}$$

where $x, y \in \{0,1\}^*$ such that $|x| = |y|$, $\lambda \in \mathbb{N}$, and $\overset{c}{\equiv}$ denotes computational indistinguishability of the ensembles for all large enough values of λ.

Remark A.1. *We note that our protocols are: i) perfecty secure in the standalone model (i.e. in a model where the protocol is executed only once), ii) have straight-line black-box simulators, i.e., only assume oracle access to the corrupt party and do not rewind. Kushilevitz et al. [51] then implies security under general concurrent composition, which is equivalent to Universal Composable (UC) security [19].*

Remark A.2. *We will sometimes prove that a protocol π UC securely realizes a functionality $\mathcal{F}$ in the $\mathcal{G}$ hybrid model. This means that the functionality $\mathcal{G}$ is available to the parties in the real-world. If ϕ is a protocol which UC securely realizes $\mathcal{G}$, then it follows by the UC composability theorem [19] that π UC securely realizes $\mathcal{F}$ in a world where calls to $\mathcal{G}$ are replaced by calls to ϕ.*

B Differential Privacy and Label Differential Privacy

B.1 Differential Privacy

Differential Privacy (DP) provides security guarantees, which are orthogonal to the security guarantees provided by secure computation. At a high level, secure computation guarantees that the computation *process* (i.e. transforming the encrypted/secret-shared input x to the output $f(x)$) leaks no additional information besides $f(x)$. On the other hand, DP guarantees that the output $f(x)$ itself leaks no "sensitive" information about x. This is formalized in Definition B.2.

Definition B.1 (Dataset and Adjacency relationship for standard DP). Let $\mathcal{X}$ denote the universe of items. A dataset D is simply an ordered subset of $\mathcal{X}$. We sometimes parse a size n dataset D as $\{x_1, \ldots, x_n\}$. We say that two datasets D and D' are *adjacent* if their size is same i.e. $|D| = |D'| = n$ and they are identical except for a difference in a single item at some arbitrary index i.e. there exists a unique $i^* \in [n]$ s.t. $x_{i^*} \in D$, $x'_{i^*} \in D'$ and $x_{i^*} \neq x'_{i^*}$.

Definition B.2 (Differential Privacy [30]). Let $\epsilon, \delta \in \mathbb{R}_{\geq 0}$. A randomized algorithm $\mathcal{A}$ taking as input a dataset is said to be (ϵ, δ)-differentially private ((ϵ, δ)-DP) if for any two adjacent datasets D and D' (Definition B.1), and for any subset S of outputs of $\mathcal{A}$, it is the case that $\Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(D') \in S] + \delta$. If $\delta = 0$, then $\mathcal{A}$ is said to be ϵ-differentially private (ϵ-DP).

B.2 Label Differential Privacy

Label Differential Privacy (Label DP) is a relaxation of the standard DP, tailored towards some machine learning tasks, where the privacy guarantee is only required w.r.t dataset labels whereas the dataset features are considered non-sensitive. To formalize this, we need to define the notion of a dataset and an adjacency relationship between datasets.

Definition B.3 (Dataset and Adjacency relationship for Label DP). Let $\mathcal{X}$ and $\mathcal{Y}$ denote the feature and label set respectively. A dataset D is simply an ordered subset of $\mathcal{X} \times \mathcal{Y}$. We sometimes parse a size n dataset D as $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. We say that two datasets D and D' are *adjacent* if their size is same i.e. $|D| = |D'| = n$ and they are identical except for a difference in a single label at some arbitrary index i.e. there exists a unique $i^* \in [n]$ s.t. $(x_{i^*}, y_{i^*}) \in D$, $(x_{i^*}, y'_{i^*}) \in D'$ and $y_{i^*} \neq y'_{i^*}$.

Now that we defined the notion of a dataset and adjacency relationship, the Label DP notion is captured in Definition B.4.

Definition B.4 (Label Differential Privacy [36]). Let $\epsilon, \delta \in \mathbb{R}_{\geq 0}$. A randomized algorithm $\mathcal{A}$ taking as input a dataset is said to be (ϵ, δ)-label differentially private ((ϵ, δ)-label DP) if for any two adjacent training datasets D and D' (Definition B.3), and for any subset S of outputs of $\mathcal{A}$, it is the case that $\Pr[\mathcal{A}(D) \in S] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(D') \in S] + \delta$. If $\delta = 0$, then $\mathcal{A}$ is said to be simply ϵ-label differentially private (ϵ-LabelDP).

C Securely Computing RR on Bins

While the RRWithPrior protocol is aimed towards adding Label DP guarantee for model training on classification tasks, Ghazi et. al.[37] recently proposed a variant called Randomized Response on Bins (RROnBins) which handles regression tasks. We describe RROnBins in Algorithm 4 which makes black-box use of a function called ComputeOptimalPhi. Since the inner workings of the function ComputeOptimalPhi will not be relevant to us, we refer the readers to [37] for its full description. This function returns a finite set $\hat{\mathcal{Y}}$ and a non-decreasing mapping $\phi : \mathcal{Y} \to \hat{\mathcal{Y}}$ for a given label set $\mathcal{Y}$, prior p, privacy parameter ϵ and loss function ℓ. At a high-level, the inverse mapping ϕ^{-1} partitions the label set $\mathcal{Y}$ into a set of finite bins (exactly $|\hat{\mathcal{Y}}|$ many bins) and then the RROnBins algorithm simply performs vanilla RR on these bins, hence the name RROnBins. Ghazi et. al.[37] show that RROnBins satisifies ϵ-DP which is captured by the following theorem.

Theorem C.1 ([37]). *For all $y_{min} \in \mathbb{R}$, $y_{max} \in \mathbb{R}$, distribution p over $\mathcal{Y} = [y_{min}, y_{max})$, $\epsilon \in \mathbb{R}_{\geq 0}$, loss function $\ell : \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$,* RROnBins *(Algorithm 4) is ϵ-DP (Definition B.2).*

Algorithm 4: RROnBins

```
Parameters: Minimum and maximum label value $y_{\min} \in \mathbb{Z}$ and
  $y_{\max} \in \mathbb{Z}$ respectively, label set $\mathcal{Y} = [y_{\min}, y_{\max}) \subset \mathbb{R}$, privacy
  parameter $\epsilon \geq 0$, loss function $\ell : \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$, prior probability
  distribution $p$ over $\mathcal{Y}$.
Inputs: Label $y \in \mathcal{Y}$.
Output: A perturbed label $\widetilde{y} \in \mathcal{Y}$.
1 $(\hat{\mathcal{Y}}, \phi) :=$ ComputeOptimalPhi$(\mathcal{Y}, p, \epsilon, \ell)$, where
    $\hat{\mathcal{Y}} \subseteq [y_{\min}, y_{\max})$ s.t. $|\hat{\mathcal{Y}}| = k$ is finite and $\phi : \mathcal{Y} \to \hat{\mathcal{Y}}$ s.t. $\phi$ is
    non-decreasing.
2 Let $\mathcal{P}_\phi = [b_0, b_1), \ldots, [b_{k-1}, b_k)$ denote a partition of $\mathcal{Y}$ induced
    by $\phi^{-1}$ and $\hat{\mathcal{Y}} = \{\hat{y}_0, \ldots, \hat{y}_{k-1}\}$ s.t. $b_0 = y_{\min}$, $b_k = y_{\max}$, and $\phi$
    is constant over each partition, i.e., for all $i \in [0, k-1]$,
    $y' \in [b_i, b_{i+1})$, we have $\phi(y') = \hat{y}_i$.
3 Let $z' \in [0, k-1]$ indicate the partition index s.t. $b_{z'} \leq y < b_{z'+1}$.
4 Define a random variable $Z$ with values in $[0, k-1]$ and let $\mathcal{D}$ be
    the induced probability distribution s.t.
      $$\Pr[Z = z] = \begin{cases} \frac{e^\epsilon}{e^\epsilon+k-1} & z = z' \\ \frac{1}{e^\epsilon+k-1} & \text{otherwise} \end{cases}$$
5 Sample $\widetilde{z} \leftarrow \mathcal{D}$ and set $\widetilde{y} = \hat{y}_{\widetilde{z}}$.
6 return $\widetilde{y}$.
```

We start by noting that our approach for securely computing RROnBins (Algorithm 4) will mirror the approach that we took in Section 4 for securely computing RRWithPrior. Since the problem formulation is same as before, we directly proceed to the next step where we simplify the sampling step in RROnBins (Algorithm 4) so that it amenable to secure computation.

Simplifying the computation. In Algorithm 5, we describe an equivalent and simplified way of computing Steps 4-6 of Algorithm 4. At a high-level, this simplification basically represents the non-uniform distribution induced on $\widetilde{y}$ in Algorithm 4 as a composition of uniform distribution and a biased bit (Bernoulli) distribution. It is easy to see that the induced probability distribution on $\widetilde{y}$ in

Functionality: $\mathcal{F}_{\text{RROnBins}}$

Public Parameters: Minimum and maximum label value $y_{\min} \in \mathbb{Z}$ and $y_{\max} \in \mathbb{Z}$ respectively, label set $\mathcal{Y} = [y_{\min}, y_{\max}) \subset \mathbb{R}$, privacy parameter $\epsilon \geq 0$, loss function $\ell : \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$, fixed-point precision $f \in \mathbb{N}$.

(1) Get a prior distribution p over $\mathcal{Y}$ as input from P_0.
(2) Get a label $y \in \mathcal{Y}$ as input from P_1.
(3) Compute $(\hat{\mathcal{Y}}, \phi) :=$ ComputeOptimalPhi$(\mathcal{Y}, p, \epsilon, \ell)$, where $\hat{\mathcal{Y}} \subseteq [y_{\min}, y_{\max})$ s.t. $|\hat{\mathcal{Y}}| = k$ is finite and $\phi : \mathcal{Y} \to \hat{\mathcal{Y}}$ s.t. ϕ is non-decreasing.
(4) Let $\mathcal{P}_\phi = [b_0, b_1), \ldots, [b_{k-1}, b_k)$ denote a partition of $\mathcal{Y}$ induced by ϕ^{-1} and $\hat{\mathcal{Y}} = \{\hat{y}_0, \ldots, \hat{y}_{k-1}\}$ s.t. $b_0 = y_{\min}$, $b_k = y_{\max}$, and ϕ is constant over each partition, i.e., for all $i \in [k]$, $y' \in [b_{i-1}, b_i)$, we have $\phi(y') = \hat{y}_i$.
(5) Let $\mathcal{P}_{\phi,\text{fix}} = [b_{0,\text{fix}}, b_{1,\text{fix}}), \ldots, [b_{k-1,\text{fix}}, b_{k,\text{fix}})$ where $b_{i,\text{fix}} = \lfloor b_i \cdot 2^f \rfloor$ for all $i \in [0, k]$.
(6) Set $q_{\text{fix}} = \left\lfloor \frac{e^\epsilon - 1}{e^\epsilon + k - 1} \cdot 2^f \right\rfloor$.
(7) Set $y_{\text{fix}} = \lfloor y \cdot 2^f \rfloor$.
(8) Let $z' \in [0, k-1]$ indicate the partition index s.t. $b_{z',\text{fix}} \leq y_{\text{fix}} < b_{z'+1,\text{fix}}$.
(9) Sample $b \leftarrow$ Bernoulli$(q_{\text{fix}}/2^f)$
(10) Sample $z \leftarrow [0, k-1]$
(11) If $b = 1$, set $\widetilde{z} := z'$. Else set $\widetilde{z} := z$
(12) Set $\widetilde{y} := \hat{y}_{\widetilde{z}}$
(13) Send $\widetilde{y}$ to P_0

Figure 7: Ideal functionality for RROnBins

Algorithm 5 is identical to that of Algorithm 4. To show this, we will calculate the probability distribution of the random variable $\widetilde{z}$ in Algorithm 5. First note that $\Pr[\widetilde{z} = z'] = \Pr[b = 1] + \Pr[b = 0] \cdot \frac{1}{k} = q + \frac{1-q}{k} = \frac{e^\epsilon}{e^\epsilon+k-1}$. Secondly, for any $u \neq z'$, we have $\Pr[\widetilde{z} = u] = \Pr[b = 0] \cdot \frac{1}{k} = \frac{1}{e^\epsilon+k-1}$. Hence, the probability distribution of $\widetilde{z}$ (resp. $\widetilde{y}$) in Algorithm 5 matches with that of $\widetilde{z}$ (resp. $\widetilde{y}$) in Algorithm 4.

Algorithm 5: Simplified sampling in RROnBins

```
1 Sample $b \leftarrow$ Bernoulli$(q)$, where $q = \frac{e^\epsilon - 1}{e^\epsilon + k - 1}$.
2 Sample $z \leftarrow [0, k-1]$
3 If $b = 1$, set $\widetilde{z} := z'$. Else set $\widetilde{z} := z$.
4 Set $\widetilde{y} := \hat{y}_{\widetilde{z}}$
5 return $\widetilde{y}$.
```

Defining the ideal functionality. Now we define an ideal functionality $\mathcal{F}_{\text{RRonBins}}$ in Figure 7 which captures RROnBins (Algorithm 4) where we replace Steps 4-6 of Algorithm 4 with the simplified steps in Algorithm 5. The security of a protocol Π_{RRonBins} realizing $\mathcal{F}_{\text{RRonBins}}$ is captured via the standard simulation based security notion which is described in Appendix A.

Protocol construction. We first note that Steps 3-6 in $\mathcal{F}_{\text{RRonBins}}$ can be computed locally by P_0 to derive the sets $\mathcal{P}_\phi, \hat{\mathcal{Y}}$ since it only depends on the prior p and public parameters. Also, Step 7 can be locally computed by P_1 since it only depends on its private label y.

The main challenge lies in computing Steps 8 - 13 which requires the P_0's sets $\mathcal{P}_\phi, \hat{\mathcal{Y}}$, along with it's size k, to be kept private (in addition to the keeping P_1's label y private). Once again, we will take a modular approach and design sub-protocols for individually computing each step, from Steps 8 - 13, securely. Fortunately, we will be able to reuse most of the sub-protocols we described in Section 4.1 including Π_{biasBit}, Π_{rand} and $\Pi^{\binom{4}{1}}_{\mathsf{mux}}$ (we will actually replace $\Pi^{\binom{4}{1}}_{\mathsf{mux}}$ with a simpler $\Pi^{\binom{2}{1}}_{\mathsf{mux}}$ as we will need to select between two values instead of four). The only new protocol that we will need is Π_{interval}. Table 5 shows the mapping between the functionality steps and their corresponding sub-protocol names. Since Step 12-13 have a trivial protocol where both parties do a public reconstruction of $\widetilde{z}$ (from Step 11) towards P_0 and then P_0 locally looks up $\hat{y}_{\widetilde{z}}$ in its set $\hat{\mathcal{Y}}$, we do not explicitly introduce a protocol for these steps.

Steps in $\mathcal{F}_{\mathsf{RRonBins}}$	Sub-protocols
8	Π_{interval}
9	Π_{biasBit}
10	Π_{rand}
11	$\Pi^{\binom{2}{1}}_{\mathsf{mux}}$

Table 5: Mapping between steps in $\mathcal{F}_{\mathsf{RRonBins}}$ and the corresponding protocols.

Protocol: Π_{interval}

Public input: Public set $S = [0, R) \subset \mathbb{N}$ where $R \in \mathbb{N}$
Private inputs: P_0 holds a list of k buckets/partitions $\mathcal{P} = \{[c_0, c_1), \ldots, [c_{k-1}, c_k)\}$ where $c_0 = 0$ and $c_k = R$. P_1 holds a value $d \in S$.
Output: $[\![z']\!]$ where $z' \in [0, k-1]$ s.t. $c_{z'} \le d < c_{z'+1}$

(1) P_0 samples $r \leftarrow \mathbb{Z}_R$.
(2) P_0 and P_1 invoke a $\mathcal{F}^{\binom{R}{1}, \mathbb{Z}_R}_{\mathsf{OT}}$ OT:
- P_0 acts as the OT sender and sets the i^{th} string (0 - indexed) as $s_i := k_i - r \pmod R$ where k_i is selected s.t. $c_{k_i} \le i < c_{k_i+1}$.
- P_1 acts the OT receiver using d as the choice index.
- P_1 retrieves s_d and outputs it.

(3) P_0 outputs r.

Figure 8: Protocol for secure interval finding.

Secure interval finding. In this protocol, we have a public set $S = [0, R) \subset \mathbb{N}$ where $R \in \mathbb{N}$. Furthermore, P_0 privately holds a list of k buckets/partitions $\mathcal{P} = \{[c_0, c_1), \ldots, [c_{k-1}, c_k)\}$ where $c_0 = 0$ and $c_k = R$ and P_1 privately holds a value $d \in S$. Both parties would like to securely compute (an additive sharing of) a partition index $z' \in [0, k-1]$ s.t. $c_{z'} \le d < c_{z'+1}$, while keeping the partition $\mathcal{P}$, along with its size k, private w.r.t. P_0 and value d private w.r.t P_1. We describe the protocol in Figure 8. The idea behind this protocol is simple: Both parties invoke $\mathcal{F}^{\binom{R}{1}, \mathbb{Z}_R}_{\mathsf{OT}}$ where P_0 acts as the OT sender and sets up the sender strings in such a way that the i^{th} sender string encodes (a masked version of) the partition number that i will fall into. When P_1, acting as the OT receiver, uses its private value d as the OT choice index, it will retrieve (a sharing of) the partition number that d falls into.

To formally prove the security of Π_{interval}, we define an ideal functionality $\mathcal{F}_{\mathsf{interval}}$ (Figure 17 in Appendix D.6) and then state that Π_{interval} securely realizes $\mathcal{F}_{\mathsf{interval}}$ in the $\mathcal{F}^{\binom{R}{1}, \mathbb{Z}_R}_{\mathsf{OT}}$ hybrid model (Theorem D.7 in Appendix D.6). In terms of efficiency, we note that the protocol simply performs a single call to the OT functionality $\mathcal{F}^{\binom{R}{1}, \mathbb{Z}_R}_{\mathsf{OT}}$ which incurs 1 round and $\lambda \log R$ bits of communication in the offline phase and 2 rounds and $(R+1)\log R$ bits of communication (across both parties) in the online phase.

Protocol: Π_{RRonBins}

Public Parameters: Minimum and maximum label value $y_{\min} \in \mathbb{Z}$ and $y_{\max} \in \mathbb{Z}$ respectively, label set $\mathcal{Y} = [y_{\min}, y_{\max}) \subset \mathbb{R}$, privacy parameter $\epsilon \ge 0$, loss function $\ell : \mathbb{R}^2 \to \mathbb{R}_{\ge 0}$, fixed-point precision $f \in \mathbb{N}$, fixed point range $R = \lfloor (y_{\max} - y_{\min}) \cdot 2^f \rfloor$
Private inputs: P_0 holds a prior distribution p over $\mathcal{Y}$. P_1 holds a label $y \in \mathcal{Y}$.
Output: P_0 outputs a perturbed label $\widetilde{y} \in \mathcal{Y}$.

(1) P_0 computes $(\hat{\mathcal{Y}}, \phi) := \mathsf{ComputeOptimalPhi}(\mathcal{Y}, p, \epsilon, \ell)$, where $\hat{\mathcal{Y}} \subseteq [y_{\min}, y_{\max})$ s.t. $|\hat{\mathcal{Y}}| = k$ is finite and $\phi : \mathcal{Y} \to \hat{\mathcal{Y}}$ s.t. ϕ is non-decreasing.
(2) P_0 sets $\mathcal{P}_\phi = \{[b_0, b_1), \ldots, [b_{k-1}, b_k)\}$ to denote a partition of $\mathcal{Y}$ induced by ϕ^{-1} and $\hat{\mathcal{Y}} = \{\hat{y}_0, \ldots, \hat{y}_{k-1}\}$ s.t. $b_0 = y_{\min}$, $b_k = y_{\max}$, and ϕ is constant over each partition, i.e., for all $i \in [k]$, $y' \in [b_{i-1}, b_i)$, we have $\phi(y') = \hat{y}_i$.
(3) P_0 sets $\mathcal{P}_{\phi,\mathsf{fix}} = \{[b_{0,\mathsf{fix}}, b_{1,\mathsf{fix}}), \ldots, [b_{k-1,\mathsf{fix}}, b_{k,\mathsf{fix}})\}$ where $b_{i,\mathsf{fix}} = \lfloor b_i \cdot 2^f \rfloor$ for all $i \in [0, k]$. P_0 sets $Q = \{[c_0, c_1), \ldots, [c_{k-1}, c_k)\}$ where $c_i = b_{i,\mathsf{fix}} - b_{0,\mathsf{fix}}$ for all $i \in [0, k]$.
(4) P_0 sets $q_{\mathsf{fix}} = \left\lfloor \frac{e^\epsilon - 1}{e^\epsilon + k - 1} \cdot 2^f \right\rfloor$.
(5) P_1 sets $y_{\mathsf{fix}} = \lfloor y \cdot 2^f \rfloor$, $y_{\min,\mathsf{fix}} = \lfloor y_{\min} \cdot 2^f \rfloor$ and $d = y_{\mathsf{fix}} - y_{\min,\mathsf{fix}}$.
(6) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{biasBit}}$, where P_0 inputs q_{fix}, and they receive shares $[\![b]\!]_0$ and $[\![b]\!]_1$ respectively.
(7) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{interval}}$, with public parameter R, where P_0 inputs Q and P_1 inputs d, and they receive shares $[\![z']\!]_0$ and $[\![z']\!]_1$ respectively.
(8) P_0 and P_1 invoke $\mathcal{F}_{\mathsf{rand}}$, with public set $\{0, \ldots, R-1\}$, where P_0 inputs $\{0, \ldots, k-1\} \subseteq \{0, \ldots, R-1\}$, and they receive shares $[\![z]\!]_0$ and $[\![z]\!]_1$ respectively.
(9) P_0 and P_1 invoke $\mathcal{F}^{\binom{2}{1}}_{\mathsf{mux}}$ using selection shares $[\![b]\!]$ and value shares $([\![z]\!], [\![z']\!])$. They finally receive a sharing of $[\![\widetilde{z}]\!]$ over $\mathbb{Z}_R$, where $\widetilde{z} = z$ if $b = 0$, and z' otherwise.
(10) P_1 sends $[\![\widetilde{z}]\!]_1$ to P_0.
(11) P_0 reconstructs $\widetilde{z} = [\![\widetilde{z}]\!]_0 + [\![\widetilde{z}]\!]_1 \pmod R$ and outputs $\widetilde{y} := \hat{y}_{\widetilde{z}}$.

Figure 9: Protocol for secure RROnBins.

Overall protocol. Figure 9 describes the overall protocol Π_{RRonBins} for realizing $\mathcal{F}_{\mathsf{RRonBins}}$. To do so, we assume that parties have access to functionalities $\mathcal{F}_*$ corresponding to the subprotocols Π_* where $* \in \{\mathsf{rand}, \mathsf{interval}, \mathsf{biasBit}, \mathsf{mux}\}$. These functionalities are described in Appendix D. The protocol Π_{RRonBins} is simple combination of the modular sub-protocols as we described earlier in Table 5 followed by a final reconstruction of the perturbed label $\widetilde{y}$ in the last step towards P_0. To formally prove the security of Π_{RRonBins}, we state that Π_{RRonBins} securely realizes $\mathcal{F}_{\mathsf{RRonBins}}$ in the $\mathcal{F}_{\mathsf{biasBit}}, \mathcal{F}_{\mathsf{interval}}, \mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{mux}}^{\binom{2}{1}}$-hybrid model.

Theorem C.2. Π_{RRonBins} *securely realizes* $\mathcal{F}_{\mathsf{RRonBins}}$ *in the* $\mathcal{F}_{\mathsf{biasBit}}$, $\mathcal{F}_{\mathsf{interval}}, \mathcal{F}_{\mathsf{rand}}, \mathcal{F}_{\mathsf{mux}}^{\binom{2}{1}}$*-hybrid model, with respect to a semi-honest adversary.*

Proof. The proof follows directly by having the simulator control the hybrid functionalities in the ideal world.

D Functionalities and Security Proofs

We now describe the functionalities underlying the protocols that we present in this work. Then we state and prove theorems stating that our protocols securely realize these functionalities in the UC model. For ease of exposition, we suppress the usage of session-ids in our functionality descriptions. Also, we will use $\mathcal{S}$ to denote the simulator (aka ideal-world adversary) in the functionalities.

D.1 Secure Membership Test

In Figure 10, we describe the ideal functionality for the secure membership test described in Section 4.1.

Functionality: $\mathcal{F}_{\text{membership}}$

Public Parameters: Set $\mathcal{Y} = \{0, \ldots, T-1\}$, where $T \in \mathbb{N}$.

(1) Get $\mathcal{Y}_{T^*} \subseteq \mathcal{Y}$ as input from P_0 where $T^* = |\mathcal{Y}_{T^*}|$.
(2) Get $y \in \mathcal{Y}$ as input from P_1.
(3) Compute a bit $b := \mathbf{1}\{y \in \mathcal{Y}_{T^*}\}$.
(4) If P_0 is corrupt, get a bit $[\![b]\!]_0$ from P_0. Else sample $[\![b]\!]_0 \leftarrow \{0,1\}$ and send it to P_0
(5) Set $[\![b]\!]_1 := b \oplus [\![b]\!]_0$. Send $[\![b]\!]_1$ to P_1

Figure 10: Functionality for secure membership test

Theorem D.1. *$\Pi_{\text{membership}}$ (Figure 5) securely realizes $\mathcal{F}_{\text{membership}}$ (Figure 10) in the $\mathcal{F}_{\text{OT}}^{\binom{T}{1},\mathbb{Z}_2}$-hybrid model, with respect to a semi-honest adversary.*

Proof. We first describe the simulator $\mathcal{S}$ for corrupt P_0. In this case, $\mathcal{S}$ simply sends corrupt P_0's input $\mathcal{Y}_{T^*}$ as input to $\mathcal{F}_{\text{membership}}$. Additionally, it internally executes $\mathcal{F}_{\text{OT}}^{\binom{T}{1},\mathbb{Z}_2}$ and receives the sender messages from corrupt P_0. Finally, it sends the output z of corrupt P_0 to $\mathcal{F}_{\text{membership}}$ as the value $[\![b]\!]_0$. This simulated distribution is identical to the real world distribution.

We now describe the simulator $\mathcal{S}$ for corrupt P_1. In this case, $\mathcal{S}$ first sends corrupt P_1's input y as input to $\mathcal{F}_{\text{membership}}$. On receiving an output $[\![b]\!]_1$ from $\mathcal{F}_{\text{membership}}$, $\mathcal{S}$ internally executes $\mathcal{F}_{\text{OT}}^{\binom{T}{1},\mathbb{Z}_2}$ and receives P_1 s input y and responds with the value $[\![b]\!]_1$. This simulated distribution is identical to the real world distribution.

D.2 Secure Sampling from a Private Set

In Figure 11, we describe the ideal functionality for the secure membership test described in Section 4.1.

Theorem D.2. *Π_{rand} (Figure 3) securely realizes $\mathcal{F}_{\text{rand}}$ (Figure 11) in the $\mathcal{F}_{\text{OT}}^{\binom{T}{1},\mathbb{Z}_T}$-hybrid model, with respect to passive corruption.*

We first show a simulator for corrupt P_0.

$\underline{\mathcal{S}_0}$:

(1) Send $\mathcal{Y}_{T^*}$ to $\mathcal{F}_{\text{rand}}$.
(2) Invoke Π_{rand} with P_0 where P_1 is simulated as per the protocol description (with null input as P_1 does not have any input in the protocol). Let $z' \in \mathbb{Z}_T$ be the output of P_0.

Functionality: $\mathcal{F}_{\text{rand}}$

Public parameters: Set $\mathcal{Y} = \{0, \ldots, T-1\}$ and arithmetic sharing domain $\mathbb{Z}_T$, where $T \in \mathbb{N}$.

(1) Get $\mathcal{Y}_{T^*} \subseteq \mathcal{Y}_T$ as input from P_0.
(2) Sample $y \leftarrow \mathcal{Y}_{T^*}$.
(3) If P_0 is corrupt, get $[\![y]\!]_0 \in \mathbb{Z}_T$ from P_0. Else sample $[\![y]\!]_0 \leftarrow \mathbb{Z}_T$ and send it to P_0.
(4) Set $[\![y]\!]_1 := y - [\![y]\!]_0 \pmod{T}$. Send $[\![y]\!]_1$ to P_1.

Figure 11: Functionality for secure sampling from a private set.

(3) Send z' to $\mathcal{F}_{\text{rand}}$.

This simulated distribution is identical to the real world distribution. We now show a simulator for corrupt P_1.

$\underline{\mathcal{S}_1}$:

(1) Receive a share $[\![y]\!]_1$ from $\mathcal{F}_{\text{rand}}$.
(2) Internally execute $\mathcal{F}_{\text{OT}}^{\binom{T}{1},\mathbb{Z}_T}$ and receive OT sender strings from P_1.
(3) Internally execute $\mathcal{F}_{\text{OT}}^{\binom{T}{1},\mathbb{Z}_T}$ and receive OT receiver choice index from P_1. Send $[\![y]\!]_1$ to P_1.

We now define a series of hybrid experiments, with changes from one hybrid to another highlighted in blue.

- Hyb_0: Real world
- Hyb_1: Same as Hyb_0 except the following change: In the second OT interaction, set the sender strings in the following way. $\forall i \neq [\![v]\!]_1, u_i := 0$. For $i = [\![v]\!]_1, u_i := d_i - z' \pmod{N}$ where $d_i := ([\![v]\!]_0 + i \pmod{T})^{\text{th}}$ label.
- Hyb_2: Same as Hyb_1 except the following change: In the second OT interaction, set the sender strings in the following way. $\forall i \neq [\![v]\!]_1, u_i := 0$. For $i = [\![v]\!]_1, u_i := d_i - z' \pmod{N}$ where $d_i \leftarrow \mathcal{Y}_{T^*}$.
- Hyb_3: Same as Hyb_2 except the following change. Get $[\![y]\!]_1$ from $\mathcal{F}_{\text{rand}}$. In the second OT interaction, set the sender strings in the following way. $\forall i \neq [\![v]\!]_1, u_i := 0$. For $i = [\![v]\!]_1, u_i := [\![y]\!]_1$.
- Hyb_4: Same as Hyb_3 except the following change: In the first OT interaction, use 0 as the choice index.

We note that the last hybrid Hyb_4 is trivially identical to the simulated distribution. We will now show that each pair of consecutive hybrids have an identical distribution.

- $\mathsf{Hyb}_0 \equiv \mathsf{Hyb}_1$: These hybrids are perfectly indistinguishable as the joint distribution of OT receiver output $u_{[\![v]\!]_1}$ from the second OT and output z' of honest P_0 is identical between Hyb_1 and Hyb_0. In both cases, $(u_{[\![v]\!]_1}, z')$ form a random arithmetic sharing of $d_{[\![v]\!]_1}$, where $v = s_{T^*-1}$ and $d_{[\![v]\!]_1}$ is the v^{th} element in $\widetilde{\mathcal{Y}}_{T^*}$.
- $\mathsf{Hyb}_1 \equiv \mathsf{Hyb}_2$: These hybrids are perfectly indistinguishable as the joint distribution of OT receiver output $u_{[\![v]\!]_1}$

from the second OT and output z' of honest P_0 is identical between Hyb_2 and Hyb_1. In Hyb_1, $(u_{[\![v]\!]_1}, z')$ form a random arithmetic sharing of $d_{[\![v]\!]_1}$, where $v = s_{T^*-1}$ and $d_{[\![v]\!]_1}$ is the v^{th} element in $\widetilde{\mathcal{Y}}_{T^*}$. In Hyb_2, $(u_{[\![v]\!]_1}, z')$ form a random arithmetic sharing of $d_{[\![v]\!]_1}$, where $d_{[\![v]\!]_1}$ is a random element from $\mathcal{Y}_{T^*}$. Note that this indistinguishability hold crucially due to the fact that $\widetilde{\mathcal{Y}}_{T^*}$ is a randomly permuted version of $\mathcal{Y}_{T^*}$.

- $\mathsf{Hyb}_2 \equiv \mathsf{Hyb}_3$: These hybrids are perfectly indistinguishable as the joint distribution of OT receiver output $u_{[\![v]\!]_1}$ from the second OT and output of honest P_0 (which is z' in Hyb_2 and $[\![y]\!]_0$ in Hyb_3) is identical between Hyb_3 and Hyb_2. In Hyb_2, $(u_{[\![v]\!]_1}, z')$ form a random arithmetic sharing of $d_{[\![v]\!]_1}$, where $d_{[\![v]\!]_1}$ is a random element from $\mathcal{Y}_{T^*}$. In Hyb_3, $(u_{[\![v]\!]_1} = [\![y]\!]_1, [\![y]\!]_0)$ form a random arithmetic sharing of y, where y is a random element from $\mathcal{Y}_{T^*}$ sampled by $\mathcal{F}_{\mathsf{rand}}$.
- $\mathsf{Hyb}_3 \equiv \mathsf{Hyb}_4$: These hybrids are perfectly indistinguishable as the joint distribution of the view of corrupt P_1 and the output of honest parties P_0 is unmodified.

D.3 Secure Bit Sampling with Private Bias

In Figure 12, we describe the ideal functionality for the secure biased bit sampling with private bias described in Section 4.1.

Theorem D.3. *Π_{biasBit} securely realizes $\mathcal{F}_{\mathsf{biasBit}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{2^f}{1},\mathbb{Z}_2}$-hybrid model, with respect to passive corruption.*

Functionality: $\mathcal{F}_{\mathsf{biasBit}}$

Public parameter: Fixed point precision $f \in \mathbb{N}$.

(1) Get $q_{\text{fix}} \in [0, 2^f)$ as input from P_0.
(2) Sample $b \leftarrow \mathsf{Bernoulli}(\frac{q_{\text{fix}}}{2^f})$.
(3) If P_0 is corrupt, get $[\![b]\!]_0$ from P_0. Else sample $[\![b]\!]_0 \leftarrow \{0,1\}$ and send it to P_0
(4) Set $[\![b]\!]_1 := b \oplus [\![b]\!]_0$. Send $[\![b]\!]_1$ to P_1

Figure 12: Functionality for secure biased bit sampling with private bias.

Proof. We first describe the simulator $\mathcal{S}$ for corrupt P_0. In this case, $\mathcal{S}$ sends P_0's input q_{fix} to $\mathcal{F}_{\mathsf{biasBit}}$. Then, it internally executes $\mathcal{F}_{\mathsf{OT}}^{\binom{2^f}{1},\mathbb{Z}_2}$ and receives the sender messages from corrupt P_0. Finally, it sends the output z of corrupt P_0 to $\mathcal{F}_{\mathsf{biasBit}}$ as the value $[\![b]\!]_0$. This simulated distribution is identical to the real world distribution.

We now describe the simulator for corrupt P_1. In this case, $\mathcal{S}$ first receives $[\![b]\!]_1$ from $\mathcal{F}_{\mathsf{biasBit}}$. Then $\mathcal{S}$ internally executes $\mathcal{F}_{\mathsf{OT}}^{\binom{2^f}{1},\mathbb{Z}_2}$ where it receives corrupt P_1's choice index $r \in [0, 2^f)$ and responds with the value $[\![b]\!]_1$. This simulated distribution is identical to the real world distribution.

D.4 Secure Multiplexer

4 choose 1 mux. In Figure 13, we describe the ideal functionality for 4-choose-1 secure multiplexer introduced in Section 4.1.4. Figure 14 describes the corresponding protocol $\Pi_{\mathsf{mux}}^{\binom{4}{1}}$ for realizing the functionality $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$. At a high-level, $\Pi_{\mathsf{mux}}^{\binom{4}{1}}$ performs two calls to $\mathcal{F}_{\mathsf{OT}}^{\binom{4}{1},\mathbb{Z}_T}$ where the first (resp. second) OT is used to generate a sharing of $[\![x^{c,d}]\!]_0$ (resp. $[\![x^{c,d}]\!]_1$) across both parties. Once parties have a sharing of $[\![x^{c,d}]\!]_0$ and $[\![x^{c,d}]\!]_1$, they can add the shares locally (due to linearity of the sharing scheme) to derive a sharing of $x^{c,d}$. We now explain the working of first OT; the second OT works in a similar way but with the sender and receiver roles reversed. In the first OT, P_0 acts as the OT sender and sets up the sender strings in such a way that when P_1, acting as the OT receiver, uses $(\langle c\rangle_1, \langle d\rangle_1)$ as the choice index, then it should retrieve (a masked version of) $[\![x^{c,d}]\!]_0$. P_0 can set up the OT sender strings in this desired fashion simply by enumerating over possible values of $(\langle c\rangle_1, \langle d\rangle_1)$ (there are 4 possible values).

Functionality: $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$

Public parameter: Arithmetic sharing domain $\mathbb{Z}_T$, where $T \in \mathbb{N}$.

(1) Get $[\![x^{0,0}]\!]_0, [\![x^{0,1}]\!]_0, [\![x^{1,0}]\!]_0, [\![x^{1,1}]\!]_0, \langle c\rangle_0, \langle d\rangle_0$ from P_0
(2) Get $[\![x^{0,0}]\!]_1, [\![x^{0,1}]\!]_1, [\![x^{1,0}]\!]_1, [\![x^{1,1}]\!]_1, \langle c\rangle_1, \langle d\rangle_1$ from P_1
(3) For all $i, j \in \{0,1\} \times \{0,1\}$, reconstruct $x^{i,j} := [\![x^{i,j}]\!]_0 + [\![x^{i,j}]\!]_1 \pmod{T}$.
(4) Compute $c := \langle c\rangle_0 \oplus \langle c\rangle_1$ and $d := \langle d\rangle_0 \oplus \langle d\rangle_1$.
(5) Set $y := x^{c,d}$.
(6) Sample $[\![y]\!]_0 \leftarrow \mathbb{Z}_T$.
(7) Set $[\![y]\!]_1 := y - [\![y]\!]_0 \pmod{\mathbb{Z}_T}$.
(8) Send $[\![y]\!]_0$ to P_0 and $[\![y]\!]_1$ to P_1.

Figure 13: Functionality for 4-choose-1 secure multiplexer.

Theorem D.4. *$\Pi_{\mathsf{mux}}^{\binom{4}{1}}$ securely realizes $\mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{4}{1},\mathbb{Z}_T}$-hybrid model, with respect to passive corruption.*

2 choose 1 mux. In Figure 15, we describe the ideal functionality for 2-choose-1 secure multiplexer. Figure 16 describes the corresponding protocol $\Pi_{\mathsf{mux}}^{\binom{2}{1}}$ for realizing the functionality $\mathcal{F}_{\mathsf{mux}}^{\binom{2}{1}}$.

Theorem D.5. *$\Pi_{\mathsf{mux}}^{\binom{2}{1}}$ securely realizes $\mathcal{F}_{\mathsf{mux}}^{\binom{2}{1}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{2}{1},\mathbb{Z}_T}$-hybrid model, with respect to passive corruption.*

D.5 Secure RRWithPrior

The security of $\Pi_{\mathsf{RRwithPrior}}$ is formally captured using the following theorem.

Theorem D.6. *$\Pi_{\mathsf{RRwithPrior}}$ (Figure 6) securely realizes $\mathcal{F}_{\mathsf{RRwithPrior}}$ (Figure 2) in the $\mathcal{F}_{\mathsf{membership}}, \mathcal{F}_{\mathsf{biasBit}}, \mathcal{F}_{\mathsf{mux}}^{\binom{4}{1}}, \mathcal{F}_{\mathsf{rand}}$ hybrid model, with respect to passive corruption.*

Protocol: $\Pi_{\text{mux}}^{\binom{4}{1}}$

Public input: Arithmetic sharing domain $\mathbb{Z}_T$, where $T \in \mathbb{N}$.
Private input: $[\![x^{0,0}]\!], [\![x^{0,1}]\!], [\![x^{1,0}]\!], [\![x^{1,1}]\!], \langle c\rangle, \langle d\rangle$.
Output: $[\![y]\!]$ where $y = x^{c,d}$.

(1) P_0 samples $r_0 \leftarrow \mathbb{Z}_T$ and P_1 samples $r_1 \leftarrow \mathbb{Z}_T$.
(2) P_0 and P_1 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{4}{1},\mathbb{Z}_T}$:
- P_0 acts as the OT sender using strings:

$$\begin{aligned}(s_0, s_1, s_2, s_3) = (&[\![x^{\langle c\rangle_0,\langle d\rangle_0}]\!]_0 - r_0,\\ &[\![x^{\langle c\rangle_0,\neg\langle d\rangle_0}]\!]_0 - r_0,\\ &[\![x^{\neg\langle c\rangle_0,\langle d\rangle_0}]\!]_0 - r_0,\\ &[\![x^{\neg\langle c\rangle_0,\neg\langle d\rangle_0}]\!]_0 - r_0)\end{aligned}$$

- P_1 acts as the OT receiver using $\langle c\rangle_1, \langle d\rangle_1$ as the choice bit.
- P_1 retrieves $y_1 := s_{2\langle c\rangle_1+\langle d\rangle_1}$.

(3) P_1 and P_0 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{4}{1},\mathbb{Z}_T}$:
- P_1 acts as the OT sender using strings:

$$\begin{aligned}(s'_0, s'_1, s'_2, s'_3) = (&[\![x^{\langle c\rangle_1,\langle d\rangle_1}]\!]_1 - r_1,\\ &[\![x^{\langle c\rangle_1,\neg\langle d\rangle_1}]\!]_1 - r_1,\\ &[\![x^{\neg\langle c\rangle_1,\langle d\rangle_1}]\!]_1 - r_1,\\ &[\![x^{\neg\langle c\rangle_1,\neg\langle d\rangle_1}]\!]_1 - r_1)\end{aligned}$$

- P_0 acts as the OT receiver using $\langle c\rangle_0, \langle d\rangle_0$ as the choice bit.
- P_0 retrieves $y_0 := s'_{2\langle c\rangle_0+\langle d\rangle_0}$.

(4) For $b \in \{0,1\}$, P_b outputs $y_b + r_b$.

Figure 14: Protocol for 4-choose-1 secure multiplexer.

Functionality: $\mathcal{F}_{\text{mux}}^{\binom{2}{1}}$

Public parameter: Arithmetic sharing domain $\mathbb{Z}_T$, where $T \in \mathbb{N}$.

(1) Get $[\![x^0]\!]_0, [\![x^1]\!]_0, \langle c\rangle_0$ from P_0
(2) Get $[\![x^0]\!]_1, [\![x^1]\!]_1, \langle c\rangle_1$ from P_1
(3) For all $i \in \{0,1\}$, reconstruct $x^i := [\![x^i]\!]_0 + [\![x^i]\!]_1 \pmod{T}$.
(4) Compute $c := \langle c\rangle_0 \oplus \langle c\rangle_1$.
(5) Set $y := x^c$.
(6) Sample $[\![y]\!]_0 \leftarrow \mathbb{Z}_T$.
(7) Set $[\![y]\!]_1 := y - [\![y]\!]_0 \pmod{\mathbb{Z}_T}$.
(8) Send $[\![y]\!]_0$ to P_0 and $[\![y]\!]_1$ to P_1.

Figure 15: Functionality for 2-choose-1 secure multiplexer.

Protocol: $\Pi_{\text{mux}}^{\binom{2}{1}}$

Public input: Arithmetic sharing domain $\mathbb{Z}_T$, where $T \in \mathbb{N}$.
Private input: $[\![x^0]\!], [\![x^1]\!], \langle c\rangle$.
Output: $[\![y]\!]$ where $y = x^c$.

(1) P_0 samples $r_0 \leftarrow \mathbb{Z}_T$ and P_1 samples $r_1 \leftarrow \mathbb{Z}_T$.
(2) P_0 and P_1 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{2}{1},\mathbb{Z}_T}$:
- P_0 acts as the OT sender using strings $(s_0, s_1) = ([\![x^{\langle c\rangle_0}]\!]_0 - r_0, [\![x^{\neg\langle c\rangle_0}]\!]_0 - r_0)$
- P_1 acts as the OT receiver using $\langle c\rangle_1$ as the choice bit.
- P_1 retrieves $y_1 := s_{\langle c\rangle_1}$.

(3) P_1 and P_0 invoke a $\mathcal{F}_{\mathsf{OT}}^{\binom{2}{1},\mathbb{Z}_T}$:
- P_1 acts as the OT sender using strings $(s'_0, s'_1) = ([\![x^{\langle c\rangle_1}]\!]_1 - r_1, [\![x^{\neg\langle c\rangle_1}]\!]_1 - r_1)$
- P_0 acts as the OT receiver using $\langle c\rangle_0$ as the choice bit.
- P_0 retrieves $y_0 := s'_{\langle c\rangle_0}$.

(4) For $b \in \{0,1\}$, P_b outputs $y_b + r_b \pmod{T}$.

Figure 16: Protocol for 2-choose-1 secure multiplexer.

Proof. We first describe the simulator $\mathcal{S}$ for corrupt P_0. Note that since the honest P_1 does not have any output, we just need to ensure that the view of corrupted P_0 in the simulated distribution is indistinguishable from its view in the real protocol. $\mathcal{S}$ starts by sending P_0's input $\vec{p}$ to $\mathcal{F}_{\mathsf{RRwithPrior}}$ and receives $\widetilde{y}$ from $\mathcal{F}_{\mathsf{RRwithPrior}}$. It then internally executes $\mathcal{F}_{\mathsf{biasBit}}, \mathcal{F}_{\mathsf{membership}}, \mathcal{F}_{\mathsf{rand}}$ where it receives $(q_{\mathsf{fix}}, [\![b_1]\!]_0), (\mathcal{Y}_{T^*}, [\![b_2]\!]_0), (\mathcal{Y}_{T^*}, [\![z]\!]_0)$ respectively from P_0. It then internally executes $\mathcal{F}_{\text{mux}}^{\binom{4}{1}}$ where it receives selection $([\![b_1]\!]_0, [\![b_2]\!]_0)$ and value shares $([\![z]\!]_0, [\![z]\!]_0, [\![z]\!]_0, [\![y]\!]_0)$ from P_0. Finally, it sends $[\![\widetilde{y}]\!]_1 := \widetilde{y} - [\![\widetilde{y}]\!]_0 \pmod{T}$ to P_0. This simulated view is identical to the real world distribution.

We now describe the simulator $\mathcal{S}$ for corrupt P_1. $\mathcal{S}$ sends P_1's input y to $\mathcal{F}_{\mathsf{RRwithPrior}}$. Then it internally executes $\mathcal{F}_{\mathsf{biasBit}}$ and $\mathcal{F}_{\mathsf{rand}}$ where it receives nothing from P_1 and sends random $[\![b_1]\!]_1$ and $[\![z]\!]_1$ respectively to P_1. $\mathcal{S}$ internally executes $\mathcal{F}_{\mathsf{membership}}$ where it receives y as input from P_1 and sends a random $[\![b_2]\!]_1$ to P_1. $\mathcal{S}$ internally executes $\mathcal{F}_{\text{mux}}^{\binom{4}{1}}$ where it receives selection $([\![b_1]\!]_1, [\![b_2]\!]_1)$ and value shares $([\![z]\!]_1, [\![z]\!]_1, [\![z]\!]_1, [\![y]\!]_1)$ from P_1. . Finally, it receives $[\![\widetilde{y}]\!]_1$ from P_1. This simulated view is identical to the real world distribution.

D.6 Secure Interval Finding

In Figure 17, we present the ideal functionality for the secure interval finding described in Section C.

Functionality: $\mathcal{F}_{\mathsf{interval}}$

Public parameter: Set $S = [0, R) \subset \mathbb{N}$, where $R \in \mathbb{N}$

(1) Get a list of k buckets/partitions $\mathcal{P} = \{[c_0, c_1), \ldots, [c_{k-1}, c_k)\}$ where each $c_i \in [0, R)$, $c_0 = 0$ and $c_k = R$ as input from P_0.
(2) Get a value $d \in [0, R)$ as input from P_1.
(3) Let $z' \in [0, k-1]$ indicate the partition index s.t. $c_{z'} \leq d < c_{z'}$.
(4) If P_0 is corrupt, get $[\![z']\!]_0 \in \mathbb{Z}_R$ as input from $\mathcal{S}$. Else sample $[\![z']\!]_0 \leftarrow \mathbb{Z}_R$ and send it to P_0.
(5) Compute $[\![z']\!]_1 := z' - [\![z']\!]_0 \pmod{R}$. Send $[\![z']\!]_1$ to P_1.

Figure 17: Functionality for secure interval finding.

Theorem D.7. *Π_{interval} securely realizes $\mathcal{F}_{\mathsf{interval}}$ in the $\mathcal{F}_{\mathsf{OT}}^{\binom{R}{1},\mathbb{Z}_R}$-hybrid model, with respect to passive corruption.*

Proof. The proof follows directly having the simulator emulate $\mathcal{F}_{\mathsf{OT}}^{\binom{R}{1},\mathbb{Z}_R}$ in the ideal world.