Lotto: Secure Participant Selection against Adversarial Servers in Federated Learning

Zhifeng Jiang and Peng Ye, *Hong Kong University of Science and Technology;* Shiqi He, *University of Michigan;* Wei Wang, *Hong Kong University of Science and Technology;* Ruichuan Chen, *Nokia Bell Labs;* Bo Li, *Hong Kong University of Science and Technology*

https://www.usenix.org/conference/usenixsecurity24/presentation/jiang-zhifeng

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Lotto: Secure Participant Selection against Adversarial Servers in Federated Learning

Zhifeng Jiang[1] Peng Ye[1] Shiqi He[2,*] Wei Wang[1] Ruichuan Chen[3] Bo Li[1]
[1]*HKUST* [2]*University of Michigan* [3]*Nokia Bell Labs*

Abstract

In Federated Learning (FL), many privacy-enhancing techniques, such as secure aggregation and distributed differential privacy, provide security guarantees under the assumption of having an *honest majority* among participants. However, an adversarial server can strategically select compromised clients to create a dishonest majority, thereby undermining the system's security guarantees. In this paper, we present Lotto, an FL system that addresses this fundamental, yet underexplored issue by providing secure participant selection in the presence of an adversarial server. Lotto supports two selection algorithms: *random* and *informed*. To ensure random selection without a trusted server, Lotto enables each client to autonomously determine their participation using *verifiable randomness*. For informed selection, which is more vulnerable to manipulation, Lotto approximates the algorithm by employing random selection within a *refined client pool*. Theoretical analysis shows that Lotto effectively aligns the proportion of server-selected compromised participants with the base rate of dishonest clients in the population. Large-scale experiments further reveal that Lotto achieves time-to-accuracy performance comparable to that of insecure selection methods.

1 Introduction

Edge devices, such as smartphones and laptops, are becoming increasingly powerful and can collaborate to build high-quality machine learning (ML) models using data collected on the devices [36]. To protect data privacy, major companies like Google and Apple have adopted *federated learning* (FL) [44] for tasks in computer vision (CV) and natural language processing (NLP) across client devices [31, 56, 66]. In FL, a server dynamically samples a small subset of clients from a large population in each training round. The sampled clients, aka *participants*, compute individual local updates using their own data and upload only the updates to the server for global aggregation, without revealing the training data [11].

However, simply keeping client data undisclosed is insufficient to preserve data privacy. Recent studies have shown that sensitive data can be leaked through individual updates during FL training [28, 64, 68, 69, 71], and it is also possible to infer a client's data from trained models [16, 46, 53, 57] by exploiting their ability to memorize information [16, 57]. To minimize data leakage, current FL systems use *secure aggregation* techniques [8, 11, 12] to ensure that adversaries only learn the aggregated updates, not individual ones. Additionally, these systems may employ *distributed differential privacy* (DP) [6, 35, 58] to limit the extent of data leakage from the aggregated update (§2.2). Since the server may not always be trusted, these privacy-enhancing techniques commonly assume that the *majority* of the participants act honestly and can jointly detect and prevent server misbehavior.

Nonetheless, this assumption may not hold in practice – an adversarial server can strategically select compromised clients with whom it colludes, resulting in a practical attack. While the overall client population is large, the number of participants selected in each round is often limited to a few hundred for efficiency reasons [11]. Consequently, even if the majority of the population is honest (e.g., 99% out of 100,000 clients), the server can still select enough adversaries to create a dishonest majority among participants.

The presence of a dishonest majority undermines the downstream privacy-enhancing techniques, leading to significant security vulnerabilities. As shown in §2.3, the privacy guarantees provided by distributed DP degrade rapidly as the dishonest cohort expands [6, 35, 58]. Additionally, secure aggregation can only tolerate a certain proportion of clients colluding with the server; exceeding this limit would enable the adversary to reconstruct a client's update [12]. It is hence crucial to prevent the server from maliciously manipulating the client selection process.

Despite its importance, this problem remains largely unaddressed in the literature. The main challenge arises from the use of a *central server* in the cross-device FL scenarios [36].

[*]Work done while the author was doing research internship at HKUST.

In the absence of efficient peer-to-peer communication across a vast array of heterogeneous devices [38, 67], the server acts as a central hub for facilitating message exchanges between clients. Since the server is untrusted, an honest client lack a reliable view of other clients, such as their availability, utility, and integrity. Consequently, it is difficult for a client to verify whether the server has correctly executed the prescribed participant selection algorithm, as this requires the true knowledge of and/or inputs from other clients.

In this paper, we present Lotto, a novel framework that enables secure participant selection in FL for the first time. Lotto aims to provide the following desired property: **among the selected participants, the proportion of the compromised ones approximately aligns with the base rate of dishonest clients in the overall population**. Lotto uses *verifiable randomness* in the selection process.

Specifically, to achieve secure random selection [44], Lotto allows each client in the population to determine its participation in a training round by computing *verifiable random functions* (VRFs) [23, 45] using its secret key and public inputs. When a client chooses to participate, Lotto collects the generated randomness and the associated VRF proof, which can be verified by other participants to ensure the integrity and validity of the randomness. This approach ensures that each client's participation is provably independent, fair, and unpredictable for other clients. However, it is still possible for an adversarial server to exploit this process and increase the number of dishonest clients by, for example, sending incorrect and/or a selectively chosen messages. To mitigate such misconduct, Lotto lets honest clients verify critical messages, adding an additional layer of security. Lotto *provably ensures* that the compromised fraction of selected participants remains close to the base rate in the population (§3.4).

Lotto also provides security guarantee for more advanced algorithms, referred to as *informed selection*, which are commonly used in FL to select the best-performing participants for optimal training efficiency [17, 37, 39, 47, 61, 70]. These algorithms rely on client-specific measurements, such as processing speed and data quality, which are difficult to verify. Instead of precisely following the selection logic of these algorithms, Lotto *approximates* them by transforming an informed selection problem into a random one. To achieve this, it lets the server *refine* the client pool by excluding a small fraction of low-quality clients based on the specified measurements. Lotto then performs secure random selection within the refined pool. Although an adversarial server may attempt to manipulate this process by excluding honest clients, the security of Lotto's random selection ensures that any advantage it gains is provably small (§3.5).

We have implemented Lotto as a library[2] (§4) and evaluated its performance with various FL tasks in a large EC2 cluster configured to emulate a cross-device scenario (§5). Compared to insecure selection algorithms, Lotto slightly increases the duration of each training round by less than 10% while achieving comparable or even better time-to-accuracy performance and inducing negligible network cost.

[2]Lotto is available at `https://github.com/SamuelGong/Lotto`.

2 Background and Motivation

In this section, we begin by providing an overview of the standard FL workflow (§2.1). We then discuss the privacy concerns and prevalent privacy-preserving approaches that critically rely on an honest majority among participants (§2.2). Following this, we highlight the privacy vulnerabilities that arise when encountering a dishonest majority (§2.3) and ultimately present the motivation for our work.

2.1 Federated Learning

Federated learning (FL) [36, 44] emerges as a new private computing paradigm that enables a large number of *clients* to collaboratively train a global model under the orchestration of a *server*. The standard FL workflow is an iterative process that comprises three steps in each training round [11]. ① Participant Selection: the server samples a subset of eligible clients from the entire *population* as participants, where the eligibility criteria may include factors such as charging status and whether being connected to an unmetered network. ② Local Training: the server distributes the global model to each participant, who then trains the model using its private data to compute a *local update*. ③ Model Aggregation: the server collects local updates from participants and computes an *aggregated update*. This step is often coupled with additional privacy-enhancing techniques (§2.2). The server then uses the aggregated update to refine the global model. As participants only expose their local updates instead of raw data to other parties, FL adheres to the data minimization principle [21, 36] mandated by many privacy regulations [60]. Consequently, FL has been deployed to facilitate a broad range of edge intelligence applications [31, 56, 66].

2.2 Enhancing Data Privacy in FL

Despite the fact that client data is not directly revealed during the FL training process, extensive research has demonstrated that it is still possible to disclose sensitive data using individual updates or trained models. Specifically, an adversary can reconstruct a client's training data from its local updates through *data extraction* [28, 64, 68, 69, 71]. For image classification tasks, even with a huge number of local gradient steps (e.g., 15k) and large models like ResNet-18 [32], state-of-the-art attackers [69] can successfully recover high-quality training images at both the pixel-level and semantic-level using a single local update. Given only an aggregated update or a trained model, the adversary can also infer whether a client's private text has been used in the training set through *membership inference* [16, 46, 53, 57].

To effectively control the exposure of clients' data against these attacks, FL systems commonly employ two privacy-enhancing techniques: secure aggregation and distributed differential privacy.

Secure Aggregation (SecAgg) [12]. SecAgg is a secure multi-party computation (SMPC) protocol that enables the server to learn only the sum of participants' updates and nothing beyond that. In SecAgg, each participant i first generates two key pairs (sk_i^1, pk_i^1), (sk_i^2, pk_i^2), where the public keys (pk_i^1, pk_i^2) are shared with others. Each pair of participants (i, j) then engage in a key agreement to derive a shared key $s_{i,j} = \mathtt{KA.Agree}(sk_i^1, pk_j^1)$. Each participant i then uses a pseudorandom number generator to derive a pairwise mask, $\boldsymbol{p}_{i,j} = \mathtt{PRNG}(s_{i,j})$, for each other participant j, and adds it to its update following $\boldsymbol{y}_i = \boldsymbol{x}_i - \sum_{j<i} \boldsymbol{p}_{i,j} + \sum_{j>i} \boldsymbol{p}_{i,j}$. As a result, after all participants' masked updates are aggregated, the pairwise masks cancel out, i.e., $\sum \boldsymbol{y}_i = \sum \boldsymbol{x}_i$.

In case of client dropout, SecAgg removes the pairwise masks of the dropped participants to ensure that the remaining pairwise masks can still cancel out when aggregated. To achieve this, each participant i secret-shares its key sk_i^1 to each other participant j using a t-out-of-n secret sharing scheme, where t is the minimum number of participants required to recover the shared key. Therefore, if participant i later drops out, a sufficient number of remaining clients ($\geq t$) can help the server reconstruct its key sk_i^1, thereby enabling the recovery of the missing pairwise masks.

A security concern remains in that the server can falsely claim the dropout of participant i to recover its pairwise masks: removing these masks, the server can uncover participant i's local update x_i. To address this concern, SecAgg introduces an additional mask $\boldsymbol{q}_i = \mathtt{PRNG}(b_i)$ where b_i is independently generated by participant i and also secret-shared with each other. The modified masking process for participant i becomes:

$$\boldsymbol{y}_i = \boldsymbol{x}_i \underbrace{- \sum_{j<i} \boldsymbol{p}_{i,j} + \sum_{j>i} \boldsymbol{p}_{i,j}}_{A} + \underbrace{\boldsymbol{q}_i}_{B}. \tag{1}$$

By requiring that the server only requests secret shares of either sk_i^1 (to recover part A if participant i drops out) or b_i (to recover part B otherwise) from other remaining participants, SecAgg ensures that part A and B cannot be removed simultaneously, and the secrecy of $\boldsymbol{x}_i$ is preserved. Subsequent work, such as SecAgg+ [8], has been proposed to reduce the runtime overhead of SecAgg and increase its scalability.

Distributed Differential Privacy (DP) [6, 35]. Simply concealing individual local updates using SecAgg is insufficient as client data can still leak through the aggregated updates. Differential privacy [22, 25] can prevent this by ensuring that no specific client's participation significantly increases the likelihood of any observed aggregated update by potential adversaries. This guarantee is captured by two parameters, ε and δ. Given any neighboring training datasets D and D' that differ only in the inclusion of a single participant's data, an aggregation procedure M is (ε, δ)-differentially private if, for any given set of output R, $\Pr[M(D) \in R] \leq e^{\varepsilon} \cdot \Pr[M(D') \in R] + \delta$. Therefore, a change in a participant's contribution yields at most a multiplicative change of e^{ε} in the probability of any output R, except with probability δ.

To achieve a privacy goal $(\varepsilon_G, \delta_G)$ in an R-round training, each aggregated update needs to be perturbed by a random noise r calibrated based on ε_G, δ_G, R, and the sensitivity of the aggregated update Δ (defined as its maximum change caused by the inclusion of a single client's data). Given the use of SecAgg, the required level of noise can be achieved without relying on the (untrusted) server by having each participant i add an even share of the noise r_i to its local update, i.e.,

$$\sum_i r_i \underbrace{\overset{(\varepsilon_G, \delta_G)\text{-DP}}{\quad}}_{R,\Delta} r. \tag{2}$$

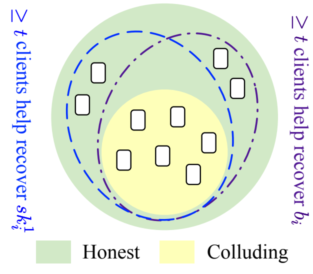
This DP model is referred to as distributed DP in the literature [36], given its distributed fashion of noise addition.

2.3 Privacy Loopholes in Common Defenses

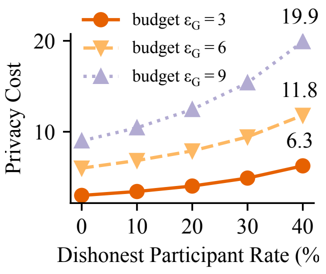
Strong Assumption on Honest Majority in Participants. The above privacy-enhancing techniques commonly assume that the *majority* of participants are honest, relying on them to detect and/or prevent the server's misbehavior. However, this assumption often proves inaccurate due to the necessity of participant selection in FL systems. While the overall population may be extensive, FL usually involves a limited number of sampled clients at each training round, typically in the range of a few hundred [11, 36]. This limitation arises because including additional clients beyond a certain threshold only yields marginal improvements in terms of convergence acceleration [44, 63]. Unfortunately, if the server acts maliciously, it can manipulate the selection process by intentionally selecting compromised clients that it colludes with, resulting in a majority of dishonest participants among those involved. Without an honest majority among participants, these techniques encounter significant security vulnerabilities.

Case #1: SecAgg. SecAgg safeguards a participant i's plaintext update $\boldsymbol{x}_i$ from server probing by distributing secret shares of sk_i^1 and b_i among other participants and requiring them to disclose only the shares of *either* secret to the server (§2.2). However, this can not be achieved if a malicious server colludes with a sufficiently large fraction of participants.

Consider a scenario with s participants, x of which collude with the server. Let t denote the threshold used by the t-out-of-n secret sharing scheme in SecAgg. As depicted in Figure 1a, when $x \geq 2t - s$, the server can identify two separate sets of honest participants, each containing at least $t - x$ members. In this case, by announcing participant i's contradictory dropout outcome to either set, the server can collect at least $t - x$ secret


≥ t clients help recover sk_i^1
≥ t clients help recover b_i
Honest
Colluding

(a) SecAgg.



(b) Distributed DP.

Figure 1: Example impacts of dishonest majority.

shares for both sk_i^1 and b_i. Combining them with the shares provided by the x colluding clients, the server can reconstruct sk_i^1 and b_i, simultaneously. Ultimately, SecAgg fails to protect the local update of participant i, as implied by Equation (1).

Case #2: Distributed DP. Distributed DP, being built upon SecAgg, inherits the aforementioned security vulnerability. When the number of dishonest participants surpasses a specific threshold ($2t - n$ as mentioned earlier), a malicious server can deduce the precise local update of participant i, which is perturbed solely by its noise share r_i (but not r).

Even if this threshold is not exceeded, the security of distributed DP can still be impacted by dishonest participants. If they choose not to add any noise to their local updates, the cumulative noise added to the aggregated update will *fall below* the required level. This results in increased data exposure, causing the system to exhaust its privacy budget more rapidly than planned. To demonstrate this issue in a practical FL scenario, we conduct an analysis on a testbed where 700 clients collaborate to train a CNN model over the FEMNIST dataset for 50 rounds. Each round involves 70 randomly selected clients, with the task of noise addition distributed equally among all participants. However, dishonest participants will abstain from adding any noise. Figure 1b illustrates that as the fraction of dishonest participants increases (while remaining below the SecAgg's threshold mentioned earlier), the privacy deficit grows explosively, regardless of the prescribed privacy budget. For example, when 40% participants are dishonest, training with a global privacy budget $\varepsilon_G = 6$ ends up consuming an ε of 11.8 at the 50th round.

3 Lotto: Secure Participant Selection

Taking the first stab at the mentioned issues, we present Lotto, a framework that secures participant selection in FL. We first formalize the participant selection problem (§3.1), establish the threat model (§3.2), and briefly introduce the cryptographic primitives used (§3.3). Next, we describe how Lotto accomplishes our goal in two scenarios: random selection (§3.4) and informed selection (§3.5). We finally provide security (§3.6) analysis for these methods.

3.1 The Participant Selection Problem

In the FL literature, participant selection is commonly approached in two forms, both of which we aim to safeguard.

Random Selection. The Federated Averaging (FedAvg) algorithm [44], considered the canonical form of FL, adapts local SGD [30, 59] on a randomly selected subset of clients during each round. To achieve random selection, the server can perform *subsampling without replacement*, where a subset of the population is chosen uniformly at random. Alternatively, it can conduct *Bernoulli sampling*, where each client is independently included with the same probability. We focus on the former as it guarantees a fixed sample size.

Informed Selection. Given the large heterogeneity in device speed and/or data characteristics, several FL advancements go beyond random sampling by employing informed selection algorithms [17, 37, 39, 47, 61, 70]. These algorithms leverage *client-specific measurements* to identify and prioritize clients who have the greatest potential to rapidly improve the global model. Among these measurements, some can be directly monitored by the server, e.g., response latencies, while the others rely on clients' self-reporting, e.g., training losses.

For example, Oort, the state-of-the-art informed algorithm, ranks each client i with a utility score U_i^{Oort} that jointly considers the client's data quality and system speed:

$$U_i^{Oort} = \underbrace{|B_i| \sqrt{\frac{1}{|B_i|} \sum_{k \in B_i} Loss(k)^2}}_{\text{Data quality}} \times \underbrace{\left(\frac{T}{t_i}\right)^{\mathbb{1}(T<t_i)\times\alpha}}_{\text{System speed}}. \quad (3)$$

The first component is the aggregate training loss that reflects the volume and distribution of the client's dataset B_i. The second component compares the client's completion time t_i with the developer-specified duration T and penalizes any delay (which makes the indicator $\mathbb{1}(T < t_i)$ outputs 1) with an exponent $\alpha > 0$. Oort prioritizes the use of clients with high utility scores for enhanced training efficiency.

Design Goal. Let n represent the population size, with c being the number of dishonest clients, and s the target number of participants in a training round. Ideally, regardless of the selection approach and the base rate of dishonest clients in the population c/n, the proportion of compromised participants x/s should match c/n, where x represents the number of compromised participants. For practical purposes, we allow for a slight relaxation of this ideal goal. Specifically, our objective is to align x/s with $\eta c/n$, where η is a small value slightly greater than 1.

3.2 Threat Model

Cross-Device Scenarios with Server-Mediated Network. Each client has a *private authenticated channel* established with the server, ensuring that their transmitted messages cannot be eavesdropped on by external parties. We do not assume a peer-to-peer network among clients due to concerns regarding efficiency and scalability. Instead, clients rely on the server to mediate communication between them.

Public Key Infrastructure (PKI). We assume a PKI that facilitates the registration of client identities and the generation of cryptographic keys on their behalf. During the setup phase, each client i *registers* its public key, denoted as (i, pk_i^{reg}), to a public bulletin board maintained by the PKI. The bulletin board only accepts registrations from clients themselves, preventing malicious parties from impersonating honest clients. Also, the PKI is responsible for *generating* the required public/secret key pair for client i to use in Lotto, $(pk_i^{Lotto}, sk_i^{Lotto})$. The PKI encrypts the secret key sk_i^{Lotto} using pk_i^{reg} to securely transmit them to client i. Meanwhile, the public key pk_i^{Lotto} is registered on the public bulletin board. The communication with the PKI is *not* mediated by the server.

Malicious Server with Colluding Clients in Population. We assume a malicious server colluding with a subset of clients in the population and they can *arbitrarily deviate* from the protocol. They may send incorrect or chosen messages to honest clients, abort or omit messages, or engage in Sybil attacks [24] to increase the portion of clients under their control. We do *not* assume a specific fraction of dishonest clients within the population to provide fundamental security. Even in the presence of a dishonest majority in the population, our objective remains to ensure that, among the selected clients, the proportion of the compromised ones aligns with the base rate of dishonest clients in the population (§3.1). In other words, client selection cannot be manipulated.

3.3 Cryptographic Primitives

Lotto uses the following cryptographic primitives.

Verifiable Random Function (VRF) [23, 45]. A VRF is a public-key PRF that provides proof that its outputs were computed correctly. It consists of the following algorithms:

- $(pk, sk) \xleftarrow{\$} \texttt{VRF.keygen}(1^\kappa)$. The key generation algorithm randomly samples a public/secret key pair (pk, sk) from the security parameter κ.
- $(\beta, \pi) \leftarrow \texttt{VRF.eval}_{sk}(x)$. The evaluation function receives a binary string x and a secret key sk and generates a random string β and the corresponding proof π.
- $\{0,1\} \leftarrow \texttt{VRF.ver}(pk, x, \beta, \pi)$. The verification function receives a public key pk and three binary strings x, β, and π, and outputs either 0 or 1.

A secure VRF should have unique provability and pseudorandomness. The former requires that for every public key pk and input x, there is a unique output β for which a proof π exists such that the verification function outputs 1. The latter requires that no adversary can distinguish a VRF output without the accompanying proof from a random string.

Signature Scheme. A signature scheme allows a message recipient to verify that the message came from a sender it knows. It consists of the following algorithms:

- $(pk, sk) \xleftarrow{\$} \texttt{SIG.gen}(1^\kappa)$. The key generation algorithm randomly samples a public/secret key pair (pk, sk) from the security parameter κ.
- $\sigma \leftarrow \texttt{SIG.sign}_{sk}(m)$. The sign function receives a binary string m and a secret key sk and generates a signature σ.
- $\{0,1\} \leftarrow \texttt{SIG.ver}(pk, m, \sigma)$. The verification function receives a public key pk and two strings m and σ, and outputs either 0 or 1.

For correctness, a signature scheme must ensure that the verification algorithm outputs 1 when presented with a correctly generated signature for any given message. For security, we require that no PPT adversary, when given a fresh honestly generated public key and access to a signature oracle, can produce a valid signature on a message on which the oracle was queried with non-negligible probability.

Pseudorandom Function (PRF) [29]. A secure $\texttt{PRF}_k(x)$ is a family of deterministic functions indexed by a key k that map an input x into an output y in such a way that y is computationally indistinguishable from the output of a truly random function with input x.

3.4 Secure Random Selection

A faithful execution of random selection naturally achieves our goal (§3.1). However, enforcing this in the presence of a malicious server poses two key challenges:

- *Correctness* (**C1**): A malicious server can deviate from random selection while falsely claiming adherence to the protocol. How can clients prevent such misbehavior?
- *Consistency* (**C2**): Even if the protocol is faithfully executed, how can we ensure that the participants in the subsequent workflow are indeed those who were selected?

Self-Sampling with Verifiable Randomness. To address **C1**, we propose a *self-sampling* approach where each individual client in the population is responsible for determining whether it should participate in a training round based on publicly available inputs. The intuition is that honest clients have a *strong incentive* to follow a reasonable selection plan due to concerns about preserving their own privacy. For security, we need to tackle the following issues simultaneously:

- *T1.1*: Ensuring that dishonest clients cannot participate without following the prescribed self-sampling protocol.
- *T1.2*: Ensuring that honest clients who choose to participate do not get arbitrarily omitted by the malicious server.

To accomplish T1.1, Lotto introduce *verifiable randomness* to the self-sampling process to allow each participant to verify the integrity of other peers' execution. Specifically, we utilize a VRF with range $[0, m)$, and assume that the FL training needs to randomly sample s out of n clients for a specific training round r. At the beginning of the round, the server first announces s, n, r to all clients in the population. Each client i then generates a random number β_i and the associated proof π_i using its private key sk_i^{Lotto} via $\beta_i, \pi_i = \texttt{VRF}_{sk_i^{Lotto}}(r)$. To achieve an expectation of s sampled clients, only those clients with $\beta_i < sm/n$ will claim to participate by uploading their β_i's and π_i's to the server, while the remaining clients abort. Next, the server takes all the surviving clients P as participants and dispatches the collected $\{\beta_i, \pi_i\}_{i \in P}$ to each of them. Finally, each participant i verifies the validity and eligibility of its peers' randomness, i.e., checking if $\texttt{VRF.val}(pk_j^{Lotto}, r, \beta_j, \pi_j) = 1$ and $\beta_j < sm/n$ for each $j \in P$. It aborts upon any failure, or refers to this set of participants throughout the remaining stages of the training round.

By pseudorandomness of VRFs (§3.3), if each sk_i^{Lotto} is uniformly random (guaranteed by the PKI as implied in §3.2), this process is pseudorandom where each client is selected equally likely with probability s/n. Also, the adversary cannot forge a random value that falls inside the range of $[0, sm/n)$ to help a dishonest client get selected. By VRF's unique provability (§3.3), if client i's original randomness generated with sk_i^{Lotto} and r falls outside the range, no eligible proof can be produced for supporting any forged β_i' to pass an honest client's verification test that takes pk_i^{Lotto} and r as inputs.

Regarding the round index r, it should be (i) unique for each round to avoid replay attacks, and (ii) consistently used by honest clients to preserve the sampling randomness. Lotto achieves (i) by allowing each client to abort if the announced r has been used previously, and also guarantees (ii) with a consistency check (as later detailed). The above design, together with the key-pair integrity guaranteed by PKIs, ensures that each participant's randomness must be honestly generated.

Over-Selection with Controlled Residual Removal. While VRFs securely achieve Bernoulli sampling, our primarily desired form of random selection is subsampling without replacement (§3.1). To achieve a fixed sample size, Lotto further employs over-selection with residual removal based on the above process. Specifically, instead of directly targeting s sampled clients, Lotto slightly increases the expected sample size by a factor $\alpha > 1$. In this case, clients identify themselves as candidate participants C when their associated β_i's satisfy $\beta_i < \alpha sm/n$. Next, the server samples s candidates out of C uniformly at random to finalize the set of participants P, if the total number of candidates is no less than the required sample

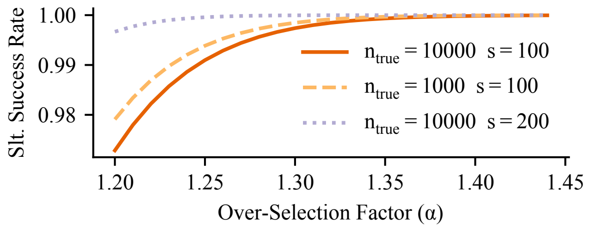


Figure 2: A small over-selection factor (α) suffices in practice.

size s (the probability of this event is given by Theorem 1) and aborts otherwise. As we observed in practice (Figure 2), $\alpha \geq 1.3$ suffices to sample enough clients with a high success rate (i.e., making Theorem 1's probability close to 1). On receiving P, clients verify whether $|P| = s$ and abort if not.

Theorem 1 (Effectiveness of Over-Selection). *Using an over-selection factor $\alpha > 0$, the described process in Lotto results in at least s candidates with probability of $1 - \sum_{i=0}^{s-1} \binom{n_{true}}{i} p^i (1-p)^{n_{true}-i}$ where $p = \alpha s/n$ and n_{true} and n is the actual size of the population and that used by clients, respectively.*

In the above process, a malicious server can intentionally remove honest candidates, not just the ones with randomly chosen indices, to gain an advantage in growing a dishonest cohort. Consider a scenario where there are c dishonest clients among a population of size n_{true}, while the server announces a population size n that may not necessarily align with n_{true}. Suppose that the candidate set C ($|C| \geq s$) contains x dishonest clients after over-selection, then the expected proportion of them in C, p_{cand}, is equal to the base rate of those in the population, i.e., $p_{cand} = \mathbb{E}\left[\frac{x}{|C|} \middle| |C| \geq s\right] = c/n_{true}$. Next, in the worst-case scenario where all the clients removed by the server are honest, the expected proportion of dishonest clients in the final s participants, p_{final}, is upper bounded as

$$
\begin{aligned}
p_{final} &= \mathbb{E}\left[\frac{x}{s} \middle| |C| \geq s\right] = \frac{1}{s}\mathbb{E}\left[x \middle| |C| \geq s\right] \\
&= \frac{1}{s} \cdot \frac{\mathbb{E}[x] - \mathbb{E}\left[x \middle| |C| < s\right] \cdot \Pr\left[|C| < s\right]}{\Pr\left[|C| \geq s\right]} \\
&< \frac{1}{s} \cdot \frac{\mathbb{E}[x]}{\Pr\left[|C| \geq s\right]} \approx \frac{1}{s}\mathbb{E}[x] = \frac{1}{s} \cdot \frac{\alpha c s}{n} = \frac{\alpha c}{n}.
\end{aligned}
$$

Thus, in expectation, the server can grow the proportion of dishonest clients by a factor approximately up to $\lambda = \frac{\alpha n_{true}}{n}$.

To ensure that λ is reasonably small (T1.2), Lotto fixes the use of α to be a small value around 1.3 since it suffices to sample enough clients as above mentioned. Lotto also ensures that n is large enough through a propose-acknowledge process. Specifically, the server proposes only once the value of n to a client during its check-in. The client then compares n with its intended number n_{min}, i.e., the population size necessary to prevent substantial inflation of the dishonest cohort from its standpoint, and proceeds with the federation only if $n \geq n_{min}$.

The Lotto Protocol for Secure Participant Selection

- **Offline Setup**:
  - All parties are given the security parameter κ, the target number of sampled clients in a round s, the over-selection factor α, a range $[0,m)$ to be used for PRF or VRF, the population threshold n_{min}, and a timeout l. All clients also have a private authenticated channel with the server and the PKI, respectively. The server knows the exact population size n.

  Client i {
  - Generate a key pair $(pk_i^{reg}, sk_i^{reg}) \leftarrow \texttt{KA.gen}(pp)$ for signature and send the public key pk_i^{reg} to the PKI for registration.
  - Receive the secret key for Lotto's use, sk_i^{Lotto}, from the PKI. Send the public key pk_i^{reg} to the server for registration.

  Server { – Receive all clients' registration keys $\{pk_i^{reg}\}$. Use them to query the PKI on the application public keys $\{pk_i^{Lotto}\}$.
- ***[Online Stage 0 (Population Refinement):]***

  Server { – Based on specific criteria, select n clients with the best utility from the population and take this set of clients as the new population.
- **Online Stage 1 (Client-Centric Selection)**:

  Server { – Announce to all clients in the population the beginning of round r and the population size n.

  Client i {
  - Upon receiving the population size n, verify that $n \geq n_{min}$. Abort if it fails.
  - Generate a random number $\beta_i, \pi_i = \texttt{VRF.eval}_{sk_i^{Lotto}}(r)$. Send to the server (β_i, π_i) and claim to join only if $\beta_i < \alpha sm/n$.

  Server { – Insert responding clients into the candidate set C and store their reported self-selection outcomes $\{(\beta_i, \pi_i)\}$ until timeout after l.
- **Online Stage 1 (Server-Centric Selection)**:

  Server { – For each client i in the population, compute $\beta_i = \texttt{PRF}_{pk_i^{Lotto}}(r)$ and insert the client into the candidate set C only if $\beta_i < \alpha sm/n$.
- **Online Stage 2 (Client Verification)**:

  Server { – If $|C| < s$, abort the current round r. Otherwise, select s clients from C uniformly at random to form the participant set $P \triangleq \{(i, pk_i^{reg}, \beta_i, \pi_i)\}$. Broadcast to all participants $i \in P$ this participant set P and the used population size n.

  Client i {
  - Receive from the server a participant set P_i and query the PKI on the public key pk_j^{Lotto} for each participant $j \in P_i$ using pk_j^{reg}.
  - Verify whether r has been used before, $(i, pk_i^{reg}, \beta_i, \pi_i) \in P_i$, $|P_i| = s$, $n \geq n_{min}$, $\beta_i < \alpha sm/n$, and $\texttt{VRF.val}(pk_j, r, \beta_j, \pi_j) = 1$ (or, $\texttt{PRF}_{pk_j^{Lotto}}(r) < \alpha sm/n$) for $\forall j \in P_i$.
  - Abort if any of the test fails. Sign the observed participant list P_i via $\sigma_i = \texttt{SIG.sign}_{sk_i^{reg}}(r||P_i)$ and send the signature σ_i to the server.
- **Online Stage 3 (Consistency Check)**:

  Server { – Collect signatures from participants in $P' \subseteq P$ and abort in case of timeout after l. Broadcast $\{j, \sigma_j\}_{j \in P'}$ to each participant $i \in P'$.

  Client i {
  - Receive from the server the set $\{j, \sigma_j\}_{j \in P'_i}$.
  - Verify $P'_i = P_i$ and $\texttt{SIG.ver}(pk_j^{reg}, r||P_i, \sigma_j) = 1$ for $\forall j \in P'_i$. Abort on any failure, or refer to P_i as participants at round r hereafter.

Figure 3: A detailed description of the Client-Centric and Server-Centric protocol for secure participant selection in Lotto (§3). *[Italicized parts inside square brackets are additionally required for informed selection.]*

Inter-Client Consistency via Signature Scheme. While we have ensured that each participant i receives a selection outcome where each member appears with upper-bounded probability, the issue of guaranteeing identical outcomes among all participants remains. To address this concern, Lotto employs a secure signature scheme, which operates as follows:

- Each client signs its received selection outcome P_i with its registration secret key sk_i^{reg}, generating a signature $\sigma_i = \texttt{SIG.sign}_{sk_i^{reg}}(r||P_i)$ where $||$ denotes concatenation.
- After collecting all signatures from the set of all responding clients P', the server broadcasts to them the set $\{(j, \sigma_j)\}_{j \in P'}$ containing the received signatures.
- Upon receiving $\{(j, \sigma_j)\}_{j \in P'_i}$, each client i verifies the correctness of all signatures and ensures that the corresponding participant lists are consistent with its observed one, i.e., $P'_i = P_i$ and $\texttt{SIG.ver}(pk_j^{reg}, r||P_i, \sigma_j) = 1$ for $\forall j \in P'_i$. If any failure occurs during verification, the client aborts.

In summary, by addressing **C1** with the above design, honest participants are ensured to proceed only with a consistent participant list P'_i of size s with each member randomly presenting with probability less than $\alpha s/n_{min}$. This upper bound remains *invariant* during the training, as it is fully determined by public values α, s, and n_{min}. Hence, this bound can be used to derive practical security guarantees (§3.6), regardless of misbehaviors exhibited by the server.

Inherent Inter-Stage Consistency with SecAgg. Ensuring consistency in participant sets used by the server throughout the entire FL workflow (**C2**) poses inherent challenges. Indeed, if the privacy-preserving protocol used in the later stage mandates that all participants observe a genuine selection outcome to ensure valid computation, **C2** is immediately addressed. *However*, if the server can secretly involve a participant set different from the announced one without being noticed by honest clients, the attainability of **C2** becomes an open problem (§6), as honest participants cannot detect such deception with limited trusted source of information (§3.2).

Fortunately, the protocol commonly employed in the remaining workflow is SecAgg (or the distributed differential privacy built upon it) (§2.2), where the server has an incen-

Table 1: Lotto's **com**putation and **com**munication complexity.

Variant	Client-Centric				Server-Centric			
	Server		Participant		Server		Participant	
	cp	cm	cp	cm	cp	cm	cp	cm
Random	$O(1)$	$O(n+s^2)$	$O(s)$		$O(n)$	$O(s^2)$	$O(s)$	
Informed	$O(n)$							

tive to uphold the actual selection outcome during the entire training. *First*, it gains no advantage by secretly involving additional dishonest clients. In SecAgg integrated with Lotto, an honest participant i establishes secure channels with peer j solely by using the public key pk_j^{reg} contained in the selection outcome P_i finalized by Lotto. Therefore, it will not share any additional secrets with clients outside of P_i, even if they participate in the halfway. *Second*, the server will not pretend arbitrarily many dropout events of selected honest participants, e.g., by omitting their messages. This is because SecAgg enforces a minimum number of participants (i.e., t) remaining in the final stage, beyond which the aggregation will abort without providing any meaningful information.

Full Protocol: Client-Centric v.s. Server-Centric. Figure 3 summarizes Lotto's overall design. We refer to this algorithm as `Client-Centric` since the selection primarily occurs at the client end. `Client-Centric` effectively achieves the security goal outlined in §3.1. We provide a detailed analysis of its security at §3.6 and its complexity in Table 1.

It is worth noting that there exists an alternative design that may initially appear to be a natural choice. This design retains the core elements of `Client-Centric` but relies on the server to perform the random selection. Specifically, for each client i in the population, the server uses a PRF with range $[0,m)$ to generate a random value $\beta_i = \mathsf{PRF}_{pk_i^{Lotto}}(r)$, and selects it as a participant if $\beta_i < \alpha sm/n$. The selection outcome, along with the associated randomness, is communicated to each participant for verification purposes. We refer to this design as `Server-Centric`. Intuitively, it offers the same security as `Client-Centric` does in a *single training round*, as it also provides honest clients with a consistent participant list with members presenting with upper-bounded probability.

However, `Server-Centric` faces a unique security challenge during *multiple-round training*. As the selection outcome relies solely on public inputs, a malicious server can accurately predict, before online training, whether an honest participant i will be selected in any round r. This opens up opportunities for the adversary to exploit the system. For instance, during residual removal after over-selection, the server can deliberately preserve the predicted most frequently appearing candidate. In the context of DP training, this coordination behavior implies a weaker level of privacy. We defer a rigorous assessment to Appendix B. Considering the similar runtime cost of the two designs (§5.2), we adopt `Client-Centric` as the first choice for its stronger security.

3.5 Secure Informed Selection

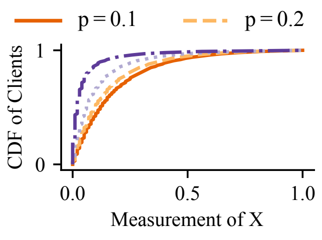
We first note that the server, despite the potential of being malicious, is unlikely to tamper with the model utility. This is because the server invests computing power in training and gains no advantage by doing otherwise. If, however, it is indeed irrational, the need for informed selection becomes unnecessary, and we can safely fall back to using Lotto's secure random selection. In both cases, Lotto aims to prevent a malicious server from manipulating the selection process for privacy attacks (§3.1).

Challenges. Unlike random selection, informed selection does not inherently incorporate randomness, but heavily relies on client measurements (§3.1). This presents two challenges in our pursuit of controlling the number of dishonest participants. Consider Oort [39] as an example, where a client's response latency and training loss are employed as proxies for data quality and system speed, respectively (§3.1). *First*, the adversary can manipulate these metrics for selecting more colluding clients. Lacking a ground truth view on these metrics, detecting such deception is hard for honest clients. *Second*, even if adversaries refrain from actively corrupting any measurements, they can still involve more dishonest clients by genuinely improving their data quality or system speed. Such exploitation is more covert to detect.

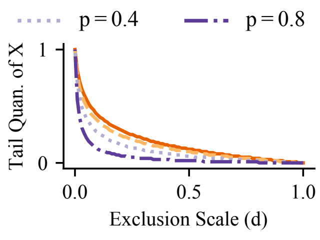
Approximation with Introduced Randomness. To sidestep the above issues, Lotto introduces randomness to the selection process by incorporating secure random selection (§3.4) to *approximate* a given informed algorithm. The server first *refines* the population by excluding a fraction d of clients with the lowest utility, based on the original criteria employed in the algorithm. It then applies secure random selection to the refined population. For example, if the original algorithm prioritizes the fastest clients, Lotto approximates it by first excluding the slowest d of clients before random selection.

Effectiveness of Population Refinement. Consider the following single-metric scenario without loss of generality: the original algorithm selects the best s clients out of the initial population of size n_{init} based on a metric X (Case A). When employing Lotto, s clients are randomly selected from the refined population of size $n = n_{init}(1-d)$ (Case B).

The extent to which Lotto approximates the original algorithm relies on the disparity in participant quality, as measured by X, between Case A and Case B. This difference, in turn, depends on the distribution of X across the initial population. Intuitively, Lotto best approximates the original algorithm when X follows a long-tailed distribution (e.g., power-law distributions [20]), where the majority of the population is considered "good" and only a small fraction is deemed "bad". In this case, Lotto can exclude almost all the relatively few "bad" participants. For example, suppose that X is the smaller, the better, and it follows Zipf's distribution parameterized by p, such that the X value of the i-th worst client is proportional



(a) X follows Zipf's distribution.



(b) Effectiveness of refinement.

Figure 4: Lotto excels in approximating the original algorithm when the majority of the initial population is "good".

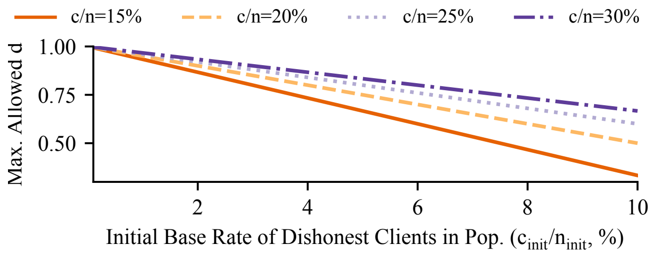


Figure 5: The maximum exclusion scale d that allows to achieve a target base rate of dishonest clients in the refined population (c/n) given various initial base rates (c_{init}/n_{init}).

to i^{-p}. In this case, precluding the worst d of the initial population enhances the quality of the worst client in the refined population by a factor of $(1-d)^{-p}$, as shown in Figure 4.

At the other extreme, there could be a scenario where the majority of clients are "bad" while only the minority are "good." In this case, our approximation may not yield participants of quality comparable to the original algorithm. We acknowledge this limitation yet contend that this compromise is a necessary cost in order to secure the selection process.

Privacy-Utility Tradeoff under a Malicious Server. While excluding more "bad" clients helps create a better-refined population, it may also exclude more honest clients. When the server behaves honestly, this does not alter the base rate of dishonest clients in the refined population c/n, since they are equally excluded. However, a malicious server can grow this rate by intentionally excluding honest clients and/or more clients than planned. To limit the impact of such misbehaviors, Lotto first enables the enforcement of a specific d by having participants check the size of the refined population n against the minimum acceptable one n_{min} (§3.4).

Next, the selection of d involves a tradeoff between privacy and utility, where a smaller d provides enhanced privacy but also reduces the potential utility. We recommend utilizing the maximum permissible exclusion scale to attain a desired base rate of dishonest clients in the refined population, while minimizing the compromise on utility, as illustrated in Figure 5. For instance, if 5% of the initial client population consists of dishonest clients, using an exclusion scale no larger than 75% can ensure that the base rate of dishonest clients in the refined population does not exceed 20% in the worst case.

Handling Multiple-Metric Algorithms. In the cases where the original algorithm incorporates two or more independent metrics, a straightforward approach to conducting population refinement is the `Or` strategy which excludes the worst clients based on one of the metrics. Alternatively, we can also adopt the `And` strategy that excludes clients that perform poorly in all metrics simultaneously. Furthermore, if the original informed algorithm combines the information from the two metrics into a single comprehensive metric, such as what Oort defines as the utility score (§3.1), there further exists a `Joint` strategy which directly relies on this single metric for population refinement. We advocate the adoption of `Or` due to its superiority demonstrated in empirical experiments (§5.3).

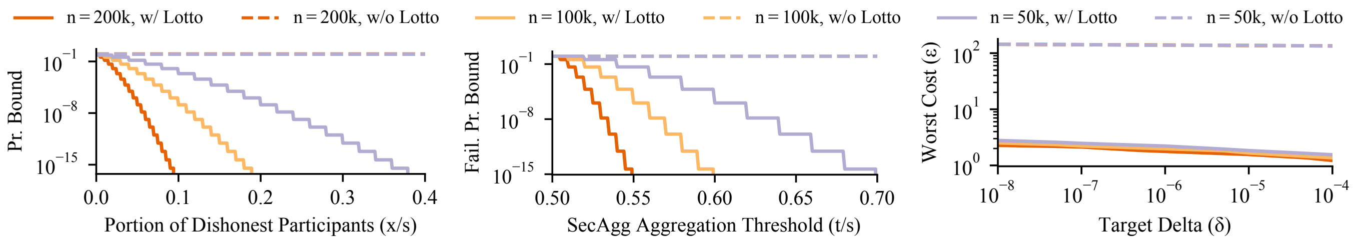
3.6 Security Analysis

Secure Random Selection. Our primary objective is to limit the number of dishonest participants (§3.1). To assess how well Lotto's random selection achieves it, we focus on the properties exhibited by the finalized participant list P. Consider a population of size n, with c clients being dishonest. We denote by s the desired number of sampled clients, and x the number of clients that an adversary could manipulate among the selected participants. Let n_{min} be the population threshold, m the randomness range, and α the over-selection factor. Theorem 2 holds despite the adversary's misbehavior.

Theorem 2 (Security of Random Selection). *If the protocol proceeds without abortion, for any $\eta > 1$, the probability that the proportion of dishonest participants, i.e., x/s, exceeds that in the population, i.e., c/n, is upper bounded as*

$$\Pr[\frac{x}{s} > \eta\frac{c}{n}] \le 1 - \sum_{i=0}^{\lfloor \eta cs/n \rfloor} \binom{c}{i} \left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{i} \left(1-\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{c-i}.$$

Intuitively, the larger the value of η, the smaller the right-hand side of the inequality mentioned above. This clearly demonstrates the effectiveness of Lotto as a secure selection framework for achieving the desired property (§3.1). While the proof is deferred to Appendix A, the significance of Theorem 2 in a practical scenario is visualized in Figure 6a. Here, we vary the population size n, while the number of colluding clients in the population set as $c = 10^3$, and the target number of participants as $s = n/1000$. We consider $\alpha = 1.3$, $m = 2^{256}$, and $n_{min} = n$. When the population size $n = 2 \times 10^5$, employing insecure random selection results in a probability of 1 for the event where the fraction of dishonest participants exceeds 5% (i.e., $\eta c/n$ with $\eta = 10$). However, with Lotto, this probability becomes negligible, approximately 1.3×10^{-7}.



(a) Direct implication. (b) End-to-end impact with SecAgg. (c) End-to-end impact with distributed DP.

Figure 6: Lotto's random selection effectively prevents arbitrary manipulation by the server in practical settings (§3.6).

We further use Theorem 2 to analyze the security of SecAgg and distributed DP (§2.2) when protected by Lotto. The related proof can also be found in Appendix A.

Corollary 1 (Bounded Failure Probability in SecAgg [12]). *Let t be the aggregation threshold of SecAgg (§2.2). With Lotto's random selection, the probability of the server being able to observe an individual update of an honest client, which indicates a failure of SecAgg, is upper bounded as*

$$\Pr[Fail] \leq 1 - \sum_{i=0}^{2t-s-1} \binom{c}{i} \left(\frac{1}{m}\left\lfloor\frac{\alpha sm}{n_{min}}\right\rfloor\right)^{i} \left(1-\frac{1}{m}\left\lfloor\frac{\alpha sm}{n_{min}}\right\rfloor\right)^{c-i}.$$

Corollary 2 (Controlled Privacy Cost in Distributed DP, I). *Let π be the distributed DP protocol parameterized by σ and built atop SecAgg with aggregation threshold t. Given target δ, an R-round FL training with π and Lotto's random selection* (`Client-Centric`) *achieves (ε,δ)-DP, where*

$$\varepsilon = \min_{\substack{0\leq k\leq \min\{c,2t-s-1\}\\ 0\leq r\leq R\\ p_k+q_r<\delta}} E_\pi\left(s,k,r,\sigma,1-\frac{1-\delta}{1-p_k-q_r}\right),$$

$$p_k = 1-\left(\sum_{i=0}^{k}\binom{c}{i}\left(\frac{1}{m}\left\lfloor\frac{\alpha sm}{n_{min}}\right\rfloor\right)^{i}\left(1-\frac{1}{m}\left\lfloor\frac{\alpha sm}{n_{min}}\right\rfloor\right)^{c-i}\right)^{R},$$

with σ the noise multiplier used in π, and $E_\pi(\cdot)$ the privacy accounting method of π. In addition, $q_r = \phi_{R,r}$ with the recurrence definition of ϕ being $\phi_{j,r} = 1-\gamma(j,r)+\gamma(j,r)\phi_{j-1,r}$ with boundary condition $\phi_{r,r} = 0$ and $\gamma(j,r) =$

$$\left(\sum_{i=0}^{r-1}\binom{j-1}{i}\left(\frac{1}{m}\left\lfloor\frac{\alpha sm}{n_{min}}\right\rfloor\right)^{i}\left(1-\frac{1}{m}\left\lfloor\frac{\alpha sm}{n_{min}}\right\rfloor\right)^{j-1-i}\right)^{s}.$$

Corollary 1 provides insights into the robustness of Lotto in maintaining the security of SecAgg, while Corollary 2 outlines the end-to-end privacy guarantees offered by Lotto when integrated with existing distributed DP systems. In the specific case discussed above, when SecAgg is used with t set larger than $0.53s$[3], the probability of SecAgg's failure diminishes to less than 8.9×10^{-6} when employing Lotto. Otherwise, the probability is 1 (Figure 6b). Building upon SecAgg with $t = 0.7s$, if DSkellam [6] is implemented as the distributed DP protocol, with a target $\delta = 1/n$ and using a consistent noise multiplier, training over FEMNIST (refer to the related settings in §5.1), Lotto guarantees a moderate privacy cost of $\varepsilon = 1.8$. In contrast, insecure selection would result in a prohibitive cost of $\varepsilon = 265$ (Figure 6c).

[3]The feasible range of t SecAgg is $(0.5s,s]$ in the malicious settings [12].

Secure Informed Selection. Population refinement in Lotto's informed selection can be deemed as a preprocessing step that precedes its random selection (§3.5). Hence, the security guarantee provided can be inherited from the random selection. Assume that the initial population consists of n_{init} clients, with c being the number of dishonest clients. Denote by n the size of the refined population. As the server can act maliciously by deliberately excluding honest clients, in the worst case, all c dishonest clients remain in the refined population. Therefore, the security guarantee can be derived by *directly* applying Theorem 2.

4 Implementation

We have implemented Lotto as a library easily pluggable into existing FL systems with 1235 lines of Python code. It operates within the server and on each client.

System Workflow. Figure 7 shows how Lotto interacts with the FL execution framework during participant selection. ① *Pool Confirmation*: the framework forms a pool with clients meeting eligibility properties (e.g., battery level, §2.1) and forward their identities and characteristics (e.g., latency) to Lotto. In case of informed selection, the server end refines the pool by excluding worst-performing clients based on specific criteria. Lotto next proceeds to random selection and has two variants. ②a *Server-Centric Selection*: the server end selects participants from the pool as candidates using PRFs. ②b *Client-Centric Selection*: for each client in the pool, the server end informs its client end to self-sample with VRFs. Clients meeting the selection conditions report their results and become candidates. ③ *Mutual Verification*: the server selects the target number of clients from candidates as par-

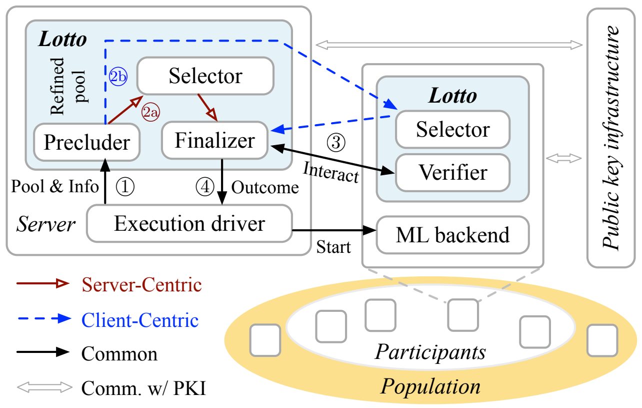


Figure 7: Lotto architecture.

ticipants. It then helps each participant verify the participant list. ④ *Finalization*: The FL framework proceeds with the participant list output by Lotto and starts FL training.

Cryptographic Instantiation. Secure PRFs can be implemented with a cryptographic hash function in conjunction with a secret key, and we use `HMAC-SHA-256`. For secure signature schemes, we utilize `Ed25519` [9], an elliptic-curve-based signature scheme. Both of the above choices are implemented leveraging the `cryptography` Python library [1]. We implement VRFs using `ECVRF-Ed25519-SHA512-Elligator2` as specified in [3] using the codebase provided by [5].

5 Evaluation

We evaluate Lotto's effectiveness in various FL training testbeds. The highlights of our evaluation are listed below.

1. Compared to insecure selection methods, Lotto introduces at most 10% runtime overhead and negligible extra network usage across different participation scales (§5.2).
2. In the case of informed selection, the adoption of Lotto achieves comparable or even superior training efficiency compared to that of insecure methods (§5.3).

5.1 Methodology

Datasets and Models. We run two common categories of applications with three real-world datasets of different scales.

- *Image Classification*: the FEMNIST [14] dataset, with 805k images classified into 62 classes, and a more complicated dataset OpenImage [2] with 1.5M images spanning 600 categories. We train a CNN [6,35], and MobileNet V2 [50] with 0.25 as width multiplier to classify the images in FEMNIST and OpenImage, respectively.
- *Language Modeling*: the large-scale Reddit dataset [4]. We train the Albert model [40] for next-word prediction.

Cluster Setup. We launch an AWS EC2 `r5n.8xlarge` instance (32 vCPUs, 256 GB memory, and 25 Gbps network) for the server and one `c5.xlarge` (4 vCPUs and 8 GB memory) instance for each client, aiming to match the computing power of mobile devices. We also throttle clients' bandwidth to fall at 44Mbps to match the average mobile bandwidth [19].

Hyperparameters. For local training, we use the mini-batch SGD for FEMNIST and OpenImage and AdamW [42] for Reddit, all with momentum set to 0.9. The number of training rounds and local epochs are 50 and 2 for both FEMNIST and Reddit and 100 and 3 for OpenImage, respectively. The batch size is consistently 20, while the learning rate is fixed at 0.01 for FEMNIST and 8e-5 for Reddit, respectively. For OpenImage, the initial learning rate is 0.05, and decreases by a factor of 0.98 after every 10 rounds. For model aggregation, we apply weighted averaging [44] protected by the SecAgg [12] protocol (§2.2).

Baselines. In evaluating random selection, we compare Lotto against Rand [44], the naive random selection protocol. As for informed selection, we compare Lotto against Oort [39], the state-of-the-art informed algorithm designed for maximum training efficiency. Both Oort and Rand are insecure as the server has the freedom to include dishonest clients. For Lotto, we primarily evaluate `Client-Centric`, while also demonstrating its similarity to `Server-Centric` (§3.4).

5.2 Execution Efficiency

We assess the time performance and network overhead of Lotto. To study the impact of the federation scale, we vary the population size from 100 to 400 to 700. The target sampled clients are proportionately set to 10, 40, and 70, respectively.

Lotto Induces Acceptable Overhead in Time. Table 2 reports the average round time measured at the server when random selection is deployed. Compared to Rand, training with `Client-Centric` incurs a time cost of less than 6%, 8%, and 10% for population sizes of 100, 400, and 700, respectively. Moreover, the overhead for `Server-Centric` is 1%, 4%, and 3%, respectively. Such acceptable overhead is expected as Lotto is built on lightweight security primitives (3.4). Table 2 also reports the time cost from the perspective of participants. In comparison to Rand, `Server-Centric` incurs less than 4% overhead at the participant side, again; whereas `Client-Centric` incurs an overhead of up to 40% due to the offloading of the client selection process to each client.[4] Regarding informed selection, the observed trends are similar to those reported in Table 2 (with Rand replaced by Oort) and are deferred to Appendix C.

[4]Such inflation barely affects the end-to-end latency (measured at the server end) as participants run in parallel with the server.

Table 2: Per-round training time and network transfer cost for the server and average participating client with random selection.

FL Application		FEMNIST@CNN				OpenImage@MobileNet				Reddit@Albert			
Population	Protocol	Time		Network		Time		Network		Time		Network	
		Server	Client	Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
100	Rand	1.76min	0.97min	64.88MB	3.9MB	3.06min	2.28min	64.35MB	3.87MB	13.0min	6.67min	958.55MB	57.46MB
	Cli-Ctr	1.86min	1.26min	64.94MB	3.9MB	3.07min	2.44min	64.4MB	3.87MB	12.86min	8.8min	958.6MB	57.46MB
	Srv-Ctr	1.77min	0.97min	64.89MB	3.9MB	2.97min	2.17min	64.36MB	3.87MB	12.88min	6.58min	958.86MB	57.46MB
400	Rand	2.56min	1.4min	0.26GB	3.56MB	4.35min	3.36min	0.25GB	3.53MB	26.94min	15.65min	3.75GB	51.53MB
	Cli-Ctr	2.59min	1.83min	0.26GB	3.56MB	4.68min	3.89min	0.25GB	3.53MB	27.53min	21.95min	3.75GB	51.53MB
	Srv-Ctr	2.29min	1.3min	0.26GB	3.56MB	4.51min	3.49min	0.25GB	3.53MB	27.17min	15.76min	3.75GB	51.53MB
700	Rand	3.46min	2.01min	0.45GB	3.69MB	5.65min	4.1min	0.45GB	3.66MB	40.06min	24.77min	6.56GB	52.57MB
	Cli-Ctr	3.82min	2.82min	0.45GB	3.69MB	6.23min	5.06min	0.45GB	3.66MB	39.59min	33.91min	6.56GB	52.57MB
	Srv-Ctr	3.56min	2.02min	0.45GB	3.7MB	5.62min	4.06min	0.45GB	3.66MB	38.85min	23.84min	6.56GB	52.57MB

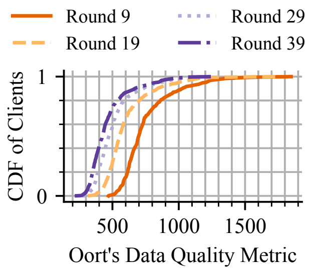
Lotto Induces Negligible Overhead in Network. Table 2 also provides the network footprint per round, measured at both the server end and the participant end. As one can see, Lotto consistently introduces less than 1% network overhead across all evaluated tasks and participation scales. This can be attributed to the fact that the predominant communication overhead in FL is the transmission of the model, rather than other auxiliary information. For example, when training over Reddit with 700 clients, model transfer in a round yields a network footprint of approximately 6.56 GB to the server. In contrast, the extra cost brought by `Client-Centric` and `Server-Centric` are merely 1.3 MB and 0.3 MB, respectively. As for informed selection, again, we defer the results to Appendix C given the similar implications.

Client-Centric v.s. Server-Centric. The preceding analysis emphasizes Lotto's minimal time and network costs, irrespective of whether it is implemented using a `Client-Centric` or `Server-Centric` approach. Given their comparable efficiency, we recommend adopting `Client-Centric`, as it ensures enhanced security (§3.4).

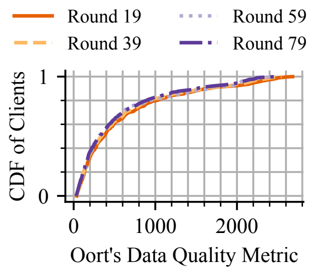
5.3 Effectiveness of Informed Selection

We next evaluate the effectiveness of Lotto's approximation strategies in achieving the objective of the target informed selection algorithm. We focus on the `Client-Centric` implementation of Lotto and use the 700-client testbed (§5.2).

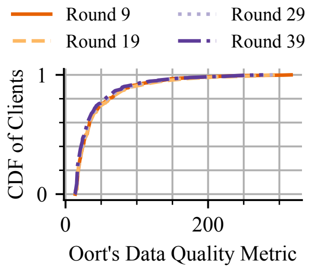
Heterogeneous Environment. The realistic datasets we employ (§5.1) already demonstrate data heterogeneity, as they are partitioned by the original data owners. Figure 8a to 8c provides a visualization of the distribution of clients' data quality, as measured by Oort's definition (the first term in Equation (3)) without loss of generality. Snapshots are taken across different rounds of the training. For Oort's considerations on system speed to be relevant, we additionally emulate hardware heterogeneity. To this end, we set the response latencies of clients to consistently follow a Zipf distribution parameterized by $a = 1.2$ (moderately skewed), regardless of the specific FL task, as illustrated in Figure 8d.



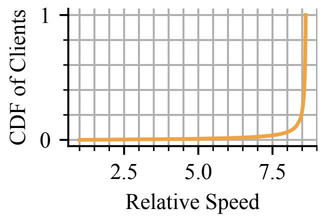
(a) FEMNIST data.



(b) OpenImage data.



(c) Reddit data.



(d) Client system speed.

Figure 8: Visualization of the heterogeneous settings.

The `Or` Strategy Outperforms Other Choices of Lotto. Lotto considers various strategies for population refinement when approximating the target informed algorithm (§3.5). Here, we compare them using a consistent exclusion threshold of 20%. Under this threshold, a client is excluded if it ranks in the bottom 20% in either system speed or data quality (`Or`); in both metrics (`And`); or in the comprehensive utility score (Equation (3)) defined by Oort (`Joint`). Additionally, we include the Rand method as a baseline reference. As illustrated in Figure 9, `Or` and `Joint` constantly outperform Rand, while `And` fails to surpass it in certain cases, e.g., FEMNIST (Fig. 9a). This can be attributed to the independent distributions of system speed and data quality across clients. As there are only a few clients that rank in the bottom 20% simultaneously for both metrics (observed to be $\leq$5% of the population), the number of clients that `And` excludes is inad-

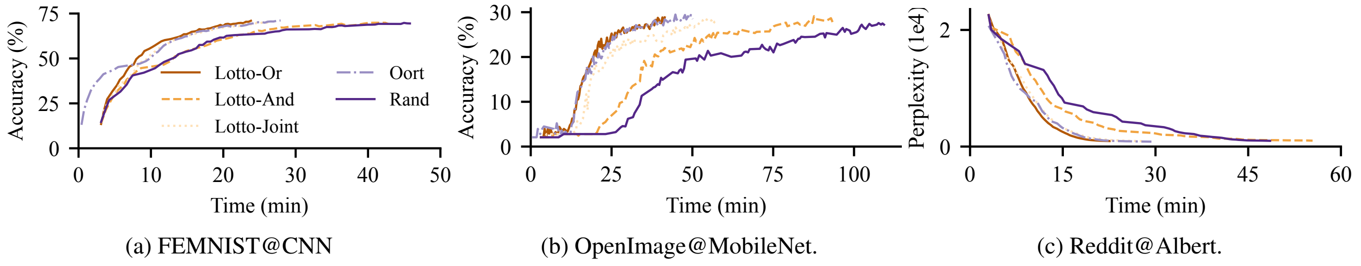


Figure 9: Lotto (with `Or` strategy) effectively approximates Oort and significantly outperforms Rand in training efficiency.

equate. Between `Or` and `Joint`, we advocate the use of the former due to its consistently superior performance and fix its use for the remaining analysis.

Lotto Achieves Comparable or Even Superior Performance to Oort. To examine the time-to-accuracy performance, we set the target accuracy at 69.8% and 27.6% for FEMNIST and OpenImage, respectively. For Reddit, we set the target perplexity (lower values are better) at 1010. As depicted in Figure 9, Lotto achieves nearly identical performance to Oort in OpenImage, and it even surpasses Oort by 1.1× for both FEMNIST and Reddit. In comparison to Rand, Lotto outperforms it by 2.1×, 3.0×, and 1.6× in FEMNIST, OpenImage, and Reddit, respectively. Note that we have accounted for the disparity in execution efficiency between Lotto and baselines (§5.2) for fairness of comparison. These results underscore the significant enhancement in the quality of selected clients through population refinement, specifically employing the `Or` strategy with a 20% exclusion factor, making the following random selection competitive with Oort which cherrypicks the best clients. Overall, the randomness introduced by Lotto for enhanced security does not compromise the training efficiency achieved by insecure methods.

6 Discussion and Future Work

Enhanced Support for SecAgg and Distributed DP. Lotto achieves a proportion of compromised participants close to the base rate of dishonest clients in the population, regardless of its magnitude (Theorem 2). Following this property, given an honest majority in the population, Lotto further ensures an honest majority among the participants. This entertains both secure aggregation (by tightening the bound in Corollary 1) and distributed differential privacy (by lowering the bound in Corollary 2). This condition can hold in practice, as simulating or compromising a large number of clients incurs prohibitive costs for the adversary given a vast client population in FL.[5] Specifically, it is expensive to register millions of identities to a PKI, which usually requires a unique, verifiable identifier for validation [8, 12]. For example, applying for Hong Kong Post eCert for individual PKI use mandates personal IDs [48]. Moreover, the costs involved in operating a client botnet at scale are also excessively high [52].

[5]If there are only tens or hundreds of clients, participant selection is no longer needed as all clients can be involved simultaneously [36].

General Solutions for Inter-Stage Consistency. Lotto does not explicitly ensure consistent reference to a participant list throughout the entire FL workflow. This is because, for SecAgg this paper primarily focuses on, any inconsistent reference by the server offers no significant advantage in probing clients' data (§3.4). This applies to other secure aggregation protocols, such as SecAgg+ [8], the state-of-the-art follow-up work of SecAgg. Similar to SecAgg, SecAgg+ ensures that each participant will not share secrets with clients beyond its verified participant list, and also includes controllable dropout tolerance that can prevent the server from pretending the dropout of numerous honest participants. However, it would be beneficial to explore general solutions for achieving inter-stage consistency when combining Lotto with other potential privacy-enhancing techniques in the future.

7 Related Work

Privacy-Preserving Aggregation with Malicious Server. Numerous protocols have been proposed to enable a server to learn the sum of multiple inputs without gaining access to individual inputs. However, many of these protocols assume an honest-but-curious server [34, 54, 55] or a trusted third party [10, 18, 26], which has limited practicality. Only the commonly used SecAgg [12] and SecAgg+ [8] and a few other protocols [41, 43, 49] allow for a malicious server. To further control the information leakage through the sum, distributed DP protocols [6, 33, 35, 58] provide rigorous privacy guarantees by combining the DP noise addition with SecAgg. However, both secure aggregation and distributed DP are still vulnerable to the privacy risks posed by a dishonest majority among participants, while Lotto takes the initiative to tackle this issue by secure participant selection.

Secure Model Utility against Adversaries. Besides probing data privacy, an adversary may also compromise the global model performance [7, 27, 51, 62]. Several approaches have been proposed to mitigate these attacks, e.g., by employing certifiable techniques during local training [13, 15, 65] and model aggregation. These approaches back up Lotto with robust model utility, whereas Lotto enhances data privacy.

8 Conclusion

Several aggregation protocols have been developed in FL to enhance the privacy of clients' data. However, the participant selection mechanisms used in these protocols are vulnerable to manipulation by adversarial servers. We proposed Lotto to address this problem. Lotto offers two secure selection algorithms, random and informed, both of which effectively align the fraction of compromised participants with the base rate of dishonest clients in the population. Theoretical analysis and large-scale experiments demonstrate that Lotto not only provides enhanced security but also achieves time-to-accuracy performance comparable to insecure selection methods, while incurring minimal network cost.

Acknowledgments

We thank anonymous reviewers for their insightful comments. We are grateful to Shaohuai Shi for providing GPU clusters to support the simulation experiments. This research was supported in part by an RGC RIF grant under the contract R6021-20, RGC CRF grants under the contracts C7004-22G and C1029-22G, and RGC GRF grants under the contracts 16209120, 16200221, 16207922, and 16211123.

References

[1] Cryptography. https://cryptography.io.

[2] Google open image dataset. https://storage.googleapis.com/openimages/web/index.html.

[3] Ietf draft-irtf-cfrg-vrf-06 specification. https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vrf-06.

[4] Reddit comment data. https://files.pushshift.io/reddit/comments.

[5] Verifiable random function codebase. https://github.com/nccgroup/draft-irtf-cfrg-vrf-06.

[6] Naman Agarwal, Peter Kairouz, and Ziyu Liu. The Skellam mechanism for differentially private federated learning. In *NeurIPS*, 2021.

[7] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *AISTATS*, 2020.

[8] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *CCS*, 2020.

[9] Daniel J Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of cryptographic engineering*, 2012.

[10] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *SOSP*, 2017.

[11] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, H Brendan McMahan, et al. Towards federated learning at scale: System design. In *MLSys*, 2019.

[12] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, 2017.

[13] Lukas Burkhalter, Hidde Lycklama, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Attestable robustness for secure federated learning. In *arXiv*, 2021.

[14] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. 2018.

[15] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Provably secure federated learning against malicious clients. In *AAAI*, 2021.

[16] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *Security*, 2019.

[17] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *HPDC*, 2020.

[18] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Eurocrypt*, 2019.

[19] Cisco. Cisco annual internet report (2018–2023) white paper, 2020. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html.

[20] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 2009.

[21] Rachel Cummings, Damien Desfontaines, David Evans, Roxana Geambasu, Matthew Jagielski, Yangsibo Huang, Peter Kairouz, Gautam Kamath, Sewoong Oh, Olga Ohrimenko, et al. Challenges towards the next frontier in privacy. In *arXiv:2304.06929*, 2023.

[22] Dwork Cynthia. Differential privacy. *Automata, languages and programming*, pages 1–12, 2006.

[23] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, 2005.

[24] John R Douceur. The sybil attack. In *IPTPS*, 2002.

[25] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[26] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *SODA*, 2019.

[27] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *Security*, 2020.

[28] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? In *NeurIPS*, 2020.

[29] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM*, 1986.

[30] Eduard Gorbunov, Filip Hanzely, and Peter Richtárik. Local sgd: Unified theory and new efficient methods. In *AISTATS*, 2021.

[31] Filip Granqvist, Matt Seigel, Rogier Van Dalen, Aine Cahill, Stephen Shum, and Matthias Paulik. Improving on-device speaker verification using federated learning with privacy. In *Interspeech*, 2020.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[33] Zhifeng Jiang, Wei Wang, and Chen Ruichuan. Efficient federated learning with dropout-resilient differential privacy. In *EuroSys*, 2024.

[34] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. In *arXiv:2009.11248*, 2020.

[35] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *ICML*, 2021.

[36] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1), 2021.

[37] Young Geun Kim and Carole-Jean Wu. Autofl: Enabling heterogeneity-aware energy efficient federated learning. In *MICRO*, 2021.

[38] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. Fedscale: Benchmarking model and system performance of federated learning at scale. In *ICML*, 2022.

[39] Fan Lai, Xiangfeng Zhu, Harsha V Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *OSDI*, 2021.

[40] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. In *ICLR*, 2020.

[41] Zizhen Liu, Si Chen, Jing Ye, Junfeng Fan, Huawei Li, and Xiaowei Li. Dhsa: efficient doubly homomorphic secure aggregation for cross-silo federated learning. *The Journal of Supercomputing*, 2023.

[42] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2018.

[43] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *S&P*, 2023.

[44] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[45] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *FOCS*, 1999.

[46] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive

and active white-box inference attacks against centralized and federated learning. In *S&P*, 2019.

[47] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC*, 2019.

[48] Hongkong Post. New e-cert application, 2024. https://www.hongkongpost.gov.hk/product/ecert/apply/certapply.html#t1.

[49] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *S&P*, 2023.

[50] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.

[51] Virat Shejwalkar and Amir Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*, 2021.

[52] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. In *S&P*, 2022.

[53] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *S&P*, 2017.

[54] Jinhyun So, Başak Güler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE JSAIT*, 2021.

[55] Jinhyun So, Chaoyang He, Chien-Sheng Yang, Songze Li, Qian Yu, Ramy E Ali, Basak Guler, and Salman Avestimehr. Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning. In *MLSys*, 2022.

[56] Congzheng Song, Filip Granqvist, and Kunal Talwar. Flair: Federated learning annotated image repository. In *NeurIPS*, 2022.

[57] Congzheng Song and Vitaly Shmatikov. Auditing data provenance in text-generation models. In *KDD*, 2019.

[58] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In *Security*, 2022.

[59] Sebastian U Stich. Local sgd converges fast and communicates little. In *ICLR*, 2018.

[60] Paul Voigt and Axel Von dem Bussche. Art. 5, ch. 2, the eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, 2017.

[61] Hao Wang, Zakhary Kaplan, Di Niu, and Baochun Li. Optimizing federated learning on non-iid data with reinforcement learning. In *INFOCOM*, 2020.

[62] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. In *NeurIPS*, 2020.

[63] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H Brendan McMahan, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, et al. A field guide to federated optimization. In *arXiv:2107.06917*, 2021.

[64] Junxiao Wang, Song Guo, Xin Xie, and Heng Qi. Protect privacy from gradient leakage attack in federated learning. In *INFOCOM*, 2022.

[65] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. Crfl: Certifiably robust federated learning against backdoor attacks. In *ICML*, 2021.

[66] Zheng Xu, Yanxiang Zhang, Galen Andrew, Christopher A Choquette-Choo, Peter Kairouz, H Brendan McMahan, Jesse Rosenstock, and Yuanbo Zhang. Federated learning of gboard language models with differential privacy. In *ACL*, 2023.

[67] Chengxu Yang, Qipeng Wang, Mengwei Xu, Zhenpeng Chen, Kaigui Bian, Yunxin Liu, and Xuanzhe Liu. Characterizing impacts of heterogeneity in federated learning upon large-scale smartphone data. In *WWW*, 2021.

[68] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *CVPR*, 2021.

[69] Kai Yue, Richeng Jin, Chau-Wai Wong, Dror Baron, and Huaiyu Dai. Gradient obfuscation gives a false sense of security in federated learning. In *Security*, 2023.

[70] Wenyu Zhang, Xiumin Wang, Pan Zhou, Weiwei Wu, and Xinglin Zhang. Client selection for federated learning with non-iid data in mobile edge computing. *IEEE Access*, 9:24462–24474, 2021.

[71] Joshua C Zhao, Atul Sharma, Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Salman Avestimehr, and Saurabh Bagchi. Secure aggregation in federated learning is not private: Leaking user data at large scale through model modification. In *arXiv:2303.12233*, 2023.

A Proofs for Security Analysis Results (§3.6)

Theorem 2. *If the protocol proceeds without abortion, for any $\eta > 1$, the probability that the proportion of dishonest participants, i.e., x/s, exceeds that in the population, i.e., c/n, is upper bounded as*

$$\Pr[\frac{x}{s} > \eta\frac{c}{n}] \leq 1 - \sum_{i=0}^{\lfloor \eta cs/n \rfloor} \binom{c}{i} \left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i \left(1 - \frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{c-i}.$$

Proof. The probability that x exceeds some value l is $\Pr[x > l] = 1 - \sum_{i=0}^{\lfloor l \rfloor}\binom{c}{i}p^i(1-p)^{c-i}$, where p is a dishonest client's chance of being selected into P. With client verification and consistency check enforced in Lotto, we have that $p = \frac{1}{m}\lfloor\frac{\alpha sm}{n}\rfloor \leq \frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor$. The theorem follows when we combine the two arguments with $l = \lfloor \eta cs/n \rfloor$. □

Corollary 1. *Let t be the aggregation threshold of SecAgg. With Lotto's random selection, the probability of the server being able to observe an individual update of an honest client, which indicates a failure of SecAgg, is upper bounded as*

$$\Pr[Fail] \leq 1 - \sum_{i=0}^{2t-s-1} \binom{c}{i} \left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i \left(1 - \frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{c-i}.$$

Proof. In SecAgg, the secrecy of an individual update is held only when the number of dishonest participants $x < 2t - s$ (see Theorem 6.5 in [12]). The corollary follows when we replace l with $2t - s - 1$ in the proof of Theorem 2. □

Corollary 2. *Let π be the distributed DP protocol parameterized by σ and built atop SecAgg with aggregation threshold t. Given target δ, an R-round FL training with π and Lotto's random selection (`Client-Centric`) achieves (ε, δ)-DP, where*

$$\varepsilon = \min_{\substack{0\leq k\leq \min\{c,2t-s-1\} \\ 0\leq r\leq R \\ p_k+q_r<\delta}} E_\pi(s,k,r,\sigma,1 - \frac{1-\delta}{1-p_k-q_r}),$$

$$p_k = 1 - \left(\sum_{i=0}^{k} \binom{c}{i} \left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i \left(1 - \frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{c-i}\right)^R,$$

with σ the noise multiplier used in π, and $E_\pi(\cdot)$ the privacy accounting method of π. In addition, $q_r = \phi_{R,r}$ with the recurrence definition of ϕ being $\phi_{j,r} = 1 - \gamma(j,r) + \gamma(j,r)\phi_{j-1,r}$ with boundary condition $\phi_{r,r} = 0$ and $\gamma(j,r) =$

$$\left(\sum_{i=0}^{r-1} \binom{j-1}{i} \left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i \left(1 - \frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{j-1-i}\right)^s.$$

Proof. In each round, each client flips a coin and becomes a candidate with probability $p = \frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor$. The server then selects s participants from the candidates according to some selection policy. First, consider the event that in each round, there are at most k colluding participants in the candidate sets (Event A). The probability of A is

$$\Pr[A] = \left(\sum_{i=0}^{k}\binom{c}{i}p^i(1-p)^{c-i}\right)^R = 1 - p_k,$$

when the FL algorithm runs for R rounds, and $p = \frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor$ given the use of Lotto (`Client-Centric`).

Next, consider the event that every client is selected in at most r rounds (Event B). We now derive a lower bound on its probability. Let $f_{n,R,r}$ be the maximum (among all possible policies) probability that at least one client is selected in at least $r+1$ rounds, given that the population has n clients and the training runs for R rounds. Consider the s clients selected by the server in the first round. Let E be the event that none of these s clients becomes a candidate in at least r rounds in the remaining $R-1$ rounds. Conditioning on E, all of them cannot be selected in at least $r+1$ rounds. Thus, if one client is selected in at least $r+1$ rounds, it must be either (i) one of the $n-s$ clients that are not selected in the first round, or (ii) selected in at least $r+1$ rounds among the rest $R-1$ rounds. We then have $f_{n,R,r} \leq \Pr[E]f_{n-s,R-1,r} + 1 - \Pr[E]$, where

$$\Pr[E] = \left(\sum_{i=0}^{r-1}\binom{R-1}{i}p^i(1-p)^{R-1-i}\right)^s.$$

By definition, $f_{n,r,r} = 0$, as no clients can be selected in $r+1$ rounds given only r rounds in total. When $n > sR$, we can see by induction that $f_{n,R,r} \leq \phi_{R,r}$, where $\phi_{R,r}$ is as defined in the corollary. For $n \leq sR$, since $f_{n,R,r}$ is non-decreasing in n (the server can simply ignore clients), we also have $f_{n,R,r} \leq f_{sR+1,R,r} \leq \phi_{R,r}$. Thus, $\Pr[B] \geq 1 - f_{n,R,r} \geq 1 - \phi_{R,r} = 1 - q_r$.

As SecAgg tolerates $2t - s - 1$ colluding clients, conditioning on the event $A \cap B$, the FL protocol is (ε', δ')-DP for $0 \leq k \leq \min\{c, 2t-s-1\}$, and $0 \leq r \leq R$, where $\varepsilon' = E_\pi(s,k,r,\sigma,\delta')$. Thus the overall FL training is $(\varepsilon', 1-(1-\delta')\Pr[A\cap B])$-DP. The union bound implies that this happens with probability $\Pr[A \cap B] \geq 1 - p_k - q_r$. Thus for a target δ, we can set $\delta' = 1 - (1-\delta)/(1-p_k-q_r)$. Minimizing ε' over all valid k and r gives the desired results. □

B Client-Centric v.s. Server-Centric

We consider `Client-Centric` as the primary approach for Lotto's random selection as it offers superior security (§3.4). We illustrate this with multiple-round DP training, where the privacy cost is contingent upon the client selected for the maximum number of times, assuming a fixed noise multiplier. With `Client-Centric`, the server cannot predict which client will be self-sampled the most frequently until the training stops. Therefore, the server cannot reserve this specific client

Table 3: Per-round training time and network transfer cost for the server and average participating client with informed selection.

FL Application		FEMNIST@CNN				OpenImage@MobileNet				Reddit@Albert			
Population	Protocol	Time		Network		Time		Network		Time		Network	
		Server	Client	Server	Client	Server	Client	Server	Client	Server	Client	Server	Client
100	Oort	1.46min	0.83min	64.88MB	3.9MB	2.7min	2.07min	64.35MB	3.87MB	12.72min	6.45min	958.55MB	57.46MB
	Cli-Ctr	1.53min	1.05min	64.97MB	3.9MB	2.84min	2.35min	64.43MB	3.87MB	12.72min	8.58min	958.63MB	57.46MB
	Srv-Ctr	1.43min	0.83min	64.89MB	3.9MB	2.76min	2.14min	64.36MB	3.87MB	12.65min	6.51min	958.56MB	57.46MB
400	Oort	2.4min	1.33min	0.26GB	3.56MB	4.44min	3.34min	0.25GB	3.53MB	27.94min	16.46min	3.75GB	51.53MB
	Cli-Ctr	2.64min	1.87min	0.26GB	3.56MB	4.69min	3.89min	0.25GB	3.54MB	28.15min	22.67min	3.75GB	51.53MB
	Srv-Ctr	2.37min	1.34min	0.26GB	3.56MB	4.43min	3.38min	0.25GB	3.53MB	27.79min	16.51min	3.75GB	51.53MB
700	Oort	3.69min	2.09min	0.45GB	3.69MB	5.71min	4.12min	0.45GB	3.66MB	40.83min	25.22min	6.56GB	52.57MB
	Cli-Ctr	3.83min	2.79min	0.46GB	3.7MB	6.01min	4.98min	0.45GB	3.67MB	40.97min	34.83min	6.56GB	52.57MB
	Srv-Ctr	3.35min	1.95min	0.45GB	3.69MB	5.37min	3.98min	0.45GB	3.66MB	39.8min	24.73min	6.56GB	52.57MB

for each round it reports to join to construct the worst-case privacy leakage. However, with `Server-Centric`, the server can predict the selection outcomes and reserve this client with certainty. Hence, `Server-Centric` provides the adversary with less randomness, as implied by the following corollary:

Corollary 3 (Controlled Privacy Cost in Distributed DP, II)**.** *Let π be the distributed DP protocol parameterized by σ and built atop SecAgg with aggregation threshold t. Given target δ, an R-round FL training with π and Lotto's random selection (`Server-Centric`) achieves (ε,δ)-DP, where*

$$\varepsilon = \min_{\substack{0\le k\le \min\{c,2t-s-1\}\\ 0\le r\le R, p_k+q_r<\delta}} E_\pi(s,k,r,\sigma,1-\frac{1-\delta}{1-p_k-q_r}),$$

$$p_k = 1-\left(\sum_{i=0}^{k}\binom{c}{i}\left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i\left(1-\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{c-i}\right)^R,$$

$$q_r = 1-\left(\sum_{i=0}^{r}\binom{R}{i}\left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i\left(1-\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{R-i}\right)^{n_{max}},$$

with σ the noise multiplier and $E_\pi(\cdot)$ the privacy accounting method for π, and n_{max} the maximum possible size of the population.

Proof. The proof closely resembles that of Corollary 2, with the exception of the calculation of $1-q_r$, which represents a lower bound on the probability that all clients in the population are selected for at most r rounds (Event B). Considering the server's predictability, we could not get a bound as tight as that of `Client-Centric`. Instead, the bound is

$$\begin{aligned}\Pr[B] &= \left(\sum_{i=0}^{r}\binom{R}{i}\left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i\left(1-\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{R-i}\right)^{n}\\ &\ge \left(\sum_{i=0}^{r}\binom{R}{i}\left(\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^i\left(1-\frac{1}{m}\lfloor\frac{\alpha sm}{n_{min}}\rfloor\right)^{R-i}\right)^{n_{max}}\\ &= 1-q_r.\end{aligned}$$

The corollary follows when one substitutes this lower bound of $\Pr[B]$ with that used in the proof of Corollary 2 and follows the remainder of that proof. □

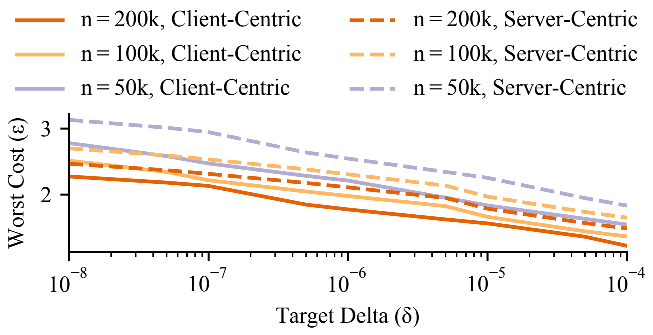


Figure 10: `Client-Centric` provides stronger DP privacy guarantees (lower ε) compared to `Server-Centric`.

To visualize the difference between the bounds presented in Corollary 2 and 3, we refer to the example discussed in §3.6. We set n_{max} to 10^9 for `Server-Centric`, which we consider to be the physical limit for the number of devices in the world. Figure 10 illustrates the gap in their privacy cost, demonstrating a slight disadvantage of `Server-Centric` across all scenarios. For instance, in the case of dealing with a population of size 2×10^5 when employing the same noise multiplier during training on FEMNIST, `Client-Centric` achieves $\varepsilon = 1.8$, whereas `Server-Centric` results in a higher cost of $\varepsilon = 2.3$.

C Execution Efficiency of Informed Selection

Table 3 presents the per-round time and network cost measured at both the server and average participant with informed selection. Compared to Oort, Lotto implemented with `Client-Centric` incurs a time cost of less than 5%, 10%, and 5% for population sizes of 100, 400, and 700, respectively. The overhead for `Server-Centric` is 2%, 0%, and 0%, respectively. Regarding the network footprint, Lotto consistently introduces less than 1% additional cost. In all, Lotto achieves an acceptable time cost and negligible network overhead in both random selection (§5.2) and informed selection.