OPA: One-shot Private Aggregation with Single Client Interaction and its Applications to Federated Learning

Harish Karthikeyan, Antigoni Polychroniadou
JPMorgan AI Research, JPMorgan AlgoCRYPT CoE
firstname.lastname@jpmchase.com

October 22, 2024

Abstract

Our work aims to minimize interaction in secure computation due to the high cost and challenges associated with communication rounds, particularly in scenarios with many clients. In this work, we revisit the problem of secure aggregation in the single-server setting where a single evaluation server can securely aggregate client-held individual inputs. Our key contribution is the introduction of One-shot Private Aggregation (OPA) where clients speak only once (or even choose not to speak) per aggregation evaluation. Since each client communicates only once per aggregation, this simplifies managing dropouts and dynamic participation, contrasting with multi-round protocols and aligning with plaintext secure aggregation, where clients interact only once.

We construct OPA based on LWR, LWE, class groups, DCR and demonstrate applications to privacy-preserving Federated Learning (FL) where clients *speak once*. This is a sharp departure from prior multi-round FL protocols whose study was initiated by Bonawitz et al. (CCS, 2017). Moreover, unlike the YOSO (You Only Speak Once) model for general secure computation, OPA eliminates complex committee selection protocols to achieve adaptive security. Beyond asymptotic improvements, OPA is practical, outperforming state-of-the-art solutions. We benchmark logistic regression classifiers for two datasets, while also building an MLP classifier to train on MNIST, CIFAR-10, and CIFAR-100 datasets.

We build two flavors of OPA (1) from (threshold) key homomorphic PRF and (2) from seed homomorphic PRG and secret sharing. The threshold Key homomorphic PRF addresses shortcomings observed in previous works that relied on DDH and LWR in the work of Boneh *et al.* (CRYPTO, 2013), marking it as an independent contribution to our work. Moreover, we also present new threshold key homomorphic PRFs based on class groups or DCR or the LWR assumption.

Contents

1 Introduction **1**

2 Our Contributions **1**
- 2.1 Overview of OPA . . . 3
- 2.2 Detailed Contributions in Federated Learning . . . 4

3 Related Work **6**
- 3.1 OPA vs Other Communication/Computation Models . . . 8

4 Technical Overview **9**

5 Preliminaries and Cryptographic Building Blocks **12**
- 5.1 Seed Homomorphic PRG . . . 12

6 One-shot Private Aggregation **13**
- 6.1 Simulation-Based Privacy . . . 14
- 6.2 Our Construction of One-shot Private Aggregation Scheme . . . 14
- 6.3 Security Against Malicious Clients . . . 18
- 6.4 One-shot Private Aggregation without Leakage Simulation . . . 19

7 Experiments **19**

A Cryptographic Preliminaries **30**
- A.1 Secret Sharing . . . 30
- A.2 Lattice-Based Assumptions . . . 34
- A.3 Pseudorandom Functions . . . 34
- A.4 Distributed Key Homomorphic PRF . . . 35
- A.5 Class Groups Framework . . . 36

B Naive Construction of OPA **37**

C Constructions in CL Framework **38**
- C.1 Distributed PRF in CL Framework . . . 40
- C.2 Construction of One-shot Private Aggregation without Leakage Simulation in CL Framework 40

D Construction from LWR **40**
- D.1 Distributed PRF from LWR . . . 40
- D.2 One-shot Private Aggregation Construction based on LWR Assumption . . . 44

E Constructions of Leakage Resilient Key-Homomorphic Pseudorandom Functions **45**
- E.1 Construction from LWR Assumption . . . 45
- E.2 Learning with Errors Assumption . . . 46

F Deferred Proofs **46**
- F.1 Proof of Theorem 9 . . . 50
- F.2 Distributed Pseudorandom Function . . . 51

G Stronger Security Definition **53**
- G.1 Updated Committee Indistinguishable Construction . . . 54

1 Introduction

Minimizing interaction in Multiparty Computation (MPC) stands as a highly sought-after objective in secure computation. This is primarily because each communication round is costly, and ensuring the liveness of participants, particularly in scenarios involving a large number of parties, poses significant challenges. Unlike throughput, latency is now primarily constrained by physical limitations, making it exceedingly difficult to substantially reduce the time required for a communication round. Furthermore, non-interactive primitives offer increased versatility and are better suited as foundational building blocks. However, any non-interactive protocol, which operates with a single communication round, becomes susceptible to a vulnerability referred to as the "residual attack" [54] where the server can collude with some clients and evaluate the function on as many inputs as they wish revealing the inputs of the honest parties.

This work explores a natural "hybrid" model that sits between the 2-round and 1-round settings. Specifically, our model allows for private aggregation, aided by a committee of members, where the client only speaks once. This approach brings us closer to achieving non-interactive protocols while preserving traditional security guarantees. Our specific focus is within the domain of secure aggregation protocols, where a group of n clients P_i for $i \in [n]$ hold a private value x_i, wish to learn the sum $\sum_i x_i$ without leaking any information about the individual x_i. In this model, clients release encoded versions of their confidential inputs x_i to a designated committee of ephemeral members and they go offline, they only speak once. Later, any subset of the ephemeral members can compute these encodings by simply transmitting a single public message to an unchanging, stateless evaluator or server. This message conveys solely the outcome of the secure aggregation and nothing else. Of significant note, the ephemeral members are stateless, speak only once, and can change (or not) per aggregation session. With that in mind, the committee members can be regarded as another subset of clients who abstain from contributing input when they are selected to serve on the committee during a current aggregation session. Each client/committee member communicates just once per aggregation, eliminating the complexity of handling dropouts commonly encountered in multi-round secure aggregation protocols. The security guarantee is that an adversary corrupting a subset of clients and the committee members learn no information about the private inputs of honest clients, beyond the sum of the outputs of these honest clients. We present a standard simulation-based security proof against both a malicious server and a semi-honest server. We present extensions that offer stronger security guarantees. See Section 3 for a comparison to other models.

Our main application is Federated Learning (FL) in which a server trains a model using data from multiple clients. This process unfolds in iterations where a randomly chosen subset of clients (or a set of clients based on the history of their availability) receives the model's current weights. These clients update the model using their local data and send back the updated weights. The server then computes the average of these weights, repeating this cycle until model convergence is achieved. This approach enhances client privacy by only accessing aggregated results, rather than raw data.

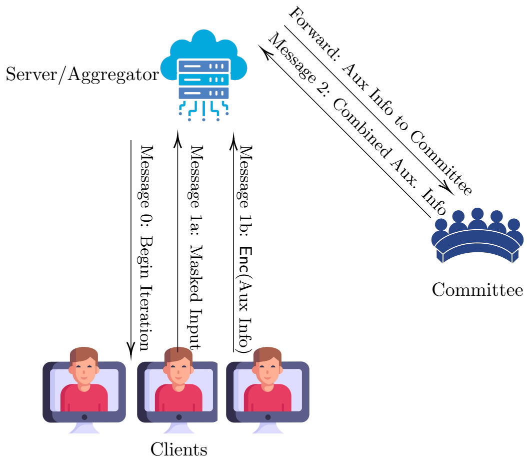
2 Our Contributions

We introduce OPA designed to achieve maximal flexibility by granting honest parties the choice to speak once or remain silent, fostering dynamic participation from one aggregation to the next. This diverges from prior approaches [17, 13, 62, 68, 52], which necessitate multiple interaction rounds and the management of dropout parties to handle communication delays or lost connections in federated learning.

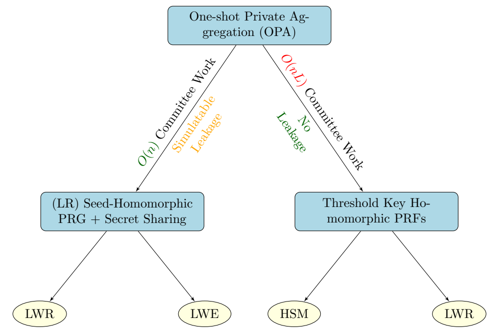
Cryptographic Assumptions: We construct OPA protocols providing a suite of six distinct versions based on a diverse spectrum of assumptions:

- Learning With Rounding (LWR) Assumption
- Learning with Errors (LWE) Assumption
- Hidden Subgroup Membership ($\mathtt{HSM}_{\mathsf{M}}$) assumption where M is a prime integer.
- $\mathtt{HSM}_{\mathsf{M}}$ assumption where $\mathsf{M} = p^k$ for some prime p and integer k.
- $\mathtt{HSM}_{\mathsf{M}}$ assumption where $\mathsf{M} = 2^k$
- $\mathtt{HSM}_{\mathsf{M}}$ assumption where $\mathsf{M} = N$ where N is an RSA modulus (i.e., the DCR assumption)

Threat Model: OPA does not require any trusted setup for keys and for M being either a prime or an exponent of prime, or the LWR assumption, we do not require any trusted setup of parameters either. We



(a) The OPA system model operates in iterations. Each iteration begins with the server sending a message to initiate the process (Message 0). In response, clients train the model on their local data, obtain updates, and mask the input. (Message 1): masked input is sent to the server, while auxiliary information is transmitted (via encryption) to the committee, via the server. Upon receiving the forwarded information, the committee members combine these into a single value. Finally, this consolidated data is sent to the server (Message 2), concluding the iteration.



(b) Our Key Contributions. Here, n is the number of clients, and L is the length of the input vector. Finally, "LR" refers to leakage-resilient. Note that the HSM assumption is parametrized by M and covers the case where M is a prime integer, $\mathsf{M} = p^k$ for some prime p and integer k, $\mathsf{M} = 2^k$, and $\mathsf{M} = N$ where N is an RSA modulus (i.e., the DCR assumption).

allow the server to be maliciously corrupted, maliciously corrupt clients, and corrupt up to a certain threshold $\mathtt{t}$ of the committee members where $\mathtt{t}$ is the corruption threshold for secret reconstruction. Based on the CL framework we additionally strengthen the security to allow for the compromise of all the committee members (while ensuring the server is not corrupt).

Variants: In Figure 1b, we summarize our variants. We offer two variants:

- OPA: A variant that uses a seed-homomorphic PRG [1] and secret sharing, with committee work independent of vector length, but leaks the sum of honest parties' keys (requiring simulatable leakage)
- OPA′: We also present a variant built from threshold/distributed key-homomorphic PRF where there is no leakage. In this case, the committee's work is dependent on the length of the input vector.

Other contributions:

- We build the first Key Homomorphic PRF based on the $\mathtt{HSM}_{\mathsf{M}}$ assumption. We later extend it to the distributed key homomorphic setting by using a modified Shamir's Secret Sharing scheme but over integers.
- We extend the almost Key Homomorphic PRF based on the LWR assumption [19, 47] using Shamir Secret Sharing over prime-order fields. In doing so, we fix gaps in the prior Distributed Key Homomorphic PRF based on LWR, as proposed by Boneh *et al.* [19]
- We also extend Shamir Secret Sharing over Integers to a packed version which enables packing more secrets in one succinct representation.
- We also present a malicious variant of OPA that empowers a server to detect a malicious client that has inconsistent shares. Meanwhile, we extend OPA′ to the setting where all the committee is corrupted but the server is honest. We want to ensure that the honest client's input is still hidden, even if the committee member gets the ciphertext.

[1] We can also build it from a (length-extended) key-homomorphic pseudorandom function (KHPRF) in the random oracle model, with the same asymptotic performance. See Section E for the instantiations

- Of independent interest, we also build a seed homomorphic PRG from the $\mathtt{HSM}_{\mathsf{M}}$ assumption for the most efficient secure aggregation protocol we present in the context of federated learning.

Applications: Our motivating application is for Federated Learning (FL). The dynamic participation feature of OPA, crucial for federated learning, facilitates secure federated learning, where participants speak only once, streamlining the process significantly. Learning in the clear involves the client receiving the global update from the server and responding with *one* message corresponding to the new updates generated after the client has trained the model with its local data. In contrast, prior works[17, 13] involve 8 rounds, and the work of [68] requires 7 rounds in total, including the setup. Moreover, our protocols offer adaptive security. Our advantages extend beyond just round complexity. See Section 2.2 and Tables 1 and 2 for a detailed comparison of asymptotics.

Implementation and Benchmarks: Our contributions also extend beyond the theoretical domain. We implement OPA as a secure aggregation protocol and benchmark with several state-of-the-art solutions [17, 13, 52, 68]. Specifically, we first implement OPA based on the CL framework with $\mathsf{M} = p$. Importantly, our server computation time scales the best with a larger number of inputs where our server takes $< 1s$ for computation even for a larger number of clients ($n = 1000$). Meanwhile, our client running time is competitive for a small number of clients but offers significant gains for a larger number of clients.

Then we benchmark OPA based on the $\mathtt{LWR}$ assumption. $\mathsf{OPA}_{\mathtt{LWR}}$, which is based on the seed homomorphic PRG, offers competitive server performance (with significant gains for larger L) when compared to $\mathsf{OPA}_{\mathsf{CL}}$. Meanwhile, $\mathsf{OPA}_{\mathtt{LWR}}$ outperforms the client performance of even $\mathsf{OPA}_{\mathsf{CL}}$. We also compare the performance of our $\mathtt{LWR}$ construction with a naive solution where each client secret-shares their inputs with the committee members. Our experiments show that $\mathsf{OPA}_{\mathtt{LWR}}$ offers the best numbers for the committee members, for large L.

To further demonstrate the feasibility of our protocol, we train a binary classification model using logistic regression, in a federated manner, for two datasets. Our protocol carefully handles floating point values (using two different methods of quantization - multiplying by a global multiplier vs representing floating point numbers as a vector of integer values) and the resulting model is shown to offer performance close to that of simply learning in the clear. We also trained a neural-network-based MLP classifier that was trained over popular machine learning datasets including MNIST, CIFAR-10, and CIFAR-100. More details can be found in Section 7.

2.1 Overview of OPA

While we defer a rigorous exposition on the technical overview of our construction to Section 4, we now present the intuition behind our protocol. The communication flow is described in Figure 1a.

Every client needs to ensure the privacy of their input, therefore a client has to mask their input. In iteration ℓ, if client i has input $\mathbf{x}_{i,\ell}$, then it chooses a mask of the same length to "add" to the inputs. Let the mask be $\mathbf{mask}_{i,\ell}$ and the ciphertext is defined as $\mathbf{ct}_{i,\ell} = \mathbf{x}_{i,\ell} + \mathbf{mask}_{i,\ell}$. To ensure privacy, we need the mask to be chosen, uniformly at random, from a large distribution. Furthermore, by performing the addition with respect to a modulus M, we get the property that for a random $\mathbf{mask}_{i,\ell}$, $\mathbf{ct}_{i,\ell}$ is identically distributed to a random element from $\mathbb{Z}_{\mathsf{M}}$. The client i sends $\mathbf{ct}_{i,\ell}$ to the server. This is Message 1a in Figure 1a.

The server, upon receiving the ciphertexts, can simply add up the ciphertexts. This leaves it with $\sum_i \mathbf{ct}_{i,\ell} = \sum_i \mathbf{x}_{i,\ell} + \sum_i \mathbf{mask}_{i,\ell}$. The goal of the server is to recover $\sum_i \mathbf{x}_{i,\ell}$. Therefore, it requires $\sum_i \mathbf{mask}_{i,\ell}$ to complete the computation. In works on Private Stream Aggregation [83, 15, 56, 47], the assumption made is that $\sum_i \mathbf{mask}_{i,\ell} = 0$. However, this requires all the clients to participate which is a difficult requirement in federated learning. Instead, we enlist the committee's help to provide the server with $\sum_i \mathbf{mask}_{i,\ell}$, for only the participating clients.

Working with the Committee. Each client i, therefore, has to communicate information about its respective $\mathbf{mask}_{i,\ell}$ to the committee members. This is what we refer to as "Aux info", and we route it through the server, to the committee member, and the information is encrypted under the public key of the respective committee member. This is Message 1b in Figure 1a. This ensures that the server cannot recover the auxiliary information. Eventually, each committee member "combines" the aux info it has received to the server (Message 2 in Figure 1a), with the guarantee that this is sufficient to reconstruct $\sum_i \mathbf{mask}_{i,\ell}$. We rely on Shamir's secret-sharing [82] to distribute a secret s to a committee of m such that as long as r number of them participate, the server can learn the secret s. The security of the secret is guaranteed even if the server

Table 1: Total asymptotic computation cost for all rounds per aggregation with semi-honest security. n denotes the total number of users, with committee size m and L is the length of the input vector. The "Rounds" column indicates the number of rounds in the setup phase (on the left, if applicable) and each aggregation iteration (on the right). A "round of communication" refers to a discrete step within a protocol during which a participant or a group of participants send messages to another participant or group of participants, and participants from the latter group must receive these messages before they can send their own messages in a subsequent round."fwd" means that the server only forwards the messages from the users. The second column in the User Aggregation phase refers to the cost of the committee members.

<table>
<tr><th rowspan="3">Protocol</th><th colspan="3" rowspan="2">Rounds</th><th colspan="5">Computation Cost</th></tr>
<tr><th colspan="2">Server</th><th colspan="3">User</th></tr>
<tr><th>Setup</th><th colspan="2">Agg.</th><th>Setup</th><th>Agg.</th><th>Setup</th><th colspan="2">Agg.</th></tr>
<tr><td>BIK+17[17]</td><td>-</td><td colspan="2">8</td><td>-</td><td>$O(n^2L)$</td><td>-</td><td colspan="2">$O(n^2 + nL)$</td></tr>
<tr><td>BBG+20[13]</td><td>-</td><td colspan="2">8</td><td>-</td><td>$O(n\log^2 n + nL\log n)$</td><td>-</td><td colspan="2">$O(\log^2 n + L\log n)$</td></tr>
<tr><td>Flamingo[68]</td><td>4</td><td colspan="2">3</td><td>fwd</td><td>$O(nL + n\log^2 n)$</td><td>$O(\log^2 n)$</td><td>$O(L + n\log n)$</td><td>$O(\mathsf{m}^2 + n)$</td></tr>
<tr><td>LERNA [62]</td><td>1</td><td>1</td><td>1</td><td>fwd</td><td>$O((\kappa + n)L + \kappa^2)$</td><td>$O(\kappa^2)$</td><td>$O(L)$</td><td>$O(L + n)$</td></tr>
<tr><td>SASH[65]</td><td>-</td><td colspan="2">10</td><td>-</td><td>$O(L + n^2)$</td><td>-</td><td colspan="2">$O(L + n^2)$</td></tr>
<tr><td>OPA</td><td>-</td><td>1</td><td>1</td><td>-</td><td>$O(nL + \mathsf{m}\log \mathsf{m})$</td><td>-</td><td>$O(L + \mathsf{m})$</td><td>$O(n)$</td></tr>
<tr><td>OPA′</td><td>-</td><td>1</td><td>1</td><td>-</td><td>$O(nL + L(\mathsf{m}\log \mathsf{m}))$</td><td>-</td><td>$O(L + \mathsf{m}L)$</td><td>$O(nL)$</td></tr>
</table>

colludes with t committee members. We require $\mathsf{t} \leqslant \mathsf{m} - 1$. We also rely on the homomorphism property which ensures that if a committee member receives shares of two secrets s_1, s_2. Then, adding up these shares will help reconstruct the secret $s_1 + s_2$.

Optimizing Committee Performance in Variant 1. The solution laid out above requires the clients to "secret-share" $\mathbf{mask}_{i,\ell}$. However, note that $\mathbf{mask}_{i,\ell}$ is as long as $\mathbf{x}_{i,\ell}$. Therefore, each committee member will receive communication $O(nL)$ where n is the number of clients and L is the length of the vector. It also has to perform computations proportional to $O(nL)$. OPA reduces the burden of the committee by introducing a succinct communication that is independent of L to the committee. We do so by relying on a structured pseudorandom generator (PRG) which is called a "seed-homomorphic PRG". A seed-homomorphic PRG ensures that $\mathsf{PRG}(\mathsf{sd}_1 + \mathsf{sd}_2) = \mathsf{PRG}(\mathsf{sd}_1) + \mathsf{PRG}(\mathsf{sd}_2)$. Now, each client i secret-shares their respective seeds $\mathsf{sd}_{i,\ell}$ (and not $\mathbf{mask}_{i,\ell}$), while setting $\mathbf{mask}_{i,\ell} = \mathsf{PRG}(\mathsf{sd}_{i,\ell})$. By the homomorphism of the secret-sharing scheme, the server can reconstruct $\sum_i \mathsf{sd}_{i,\ell}$, and then expand it as $\mathsf{PRG}(\sum_i \mathsf{sd}_{i,\ell})$. By seed-homomorphism, this is also equal to $\sum_i \mathsf{PRG}(\mathsf{sd}_{i,\ell}) = \sum_i \mathbf{mask}_{i,\ell}$.

2.2 Detailed Contributions in Federated Learning

Next, we compare our protocol with all efficient summation protocols listed in Table 3, with a specific focus on those that accommodate dynamic participation, a key feature shared by all federated learning methodologies. Asymptotically, we look at the performance of OPA. In Tables 1 and 2, we list the communication complexity, computational complexity, and round complexity per participant. Notably, our protocols are setup-free, eliminating any need for elaborate initialization procedures. Furthermore, they are characterized by a streamlined communication process, demanding just a single round of interaction from the participants.

Asymptotic Comparison. More concretely, based on Table 1, our approach stands out by significantly reducing the round complexity, ensuring that each participant's involvement is limited to a single communication round i.e. each participant speaks only once. That is, users speak once and committee members speak once too. On the contrary previous works[17, 13][2] require 8 rounds and the work of [68] requires 7 rounds in total, including the setup. This reduction in round complexity serves as a significant efficiency advantage.

Despite our advantage in the round complexity, our advantages extend beyond just round complexity (see Table 1). Notably, as the number of participants (n) grows larger, our protocol excels in terms of computational complexity. While previous solutions exhibit complexities that are quadratic [17, 65] or linearithmic [68]

[2] [13] offer a weaker security definition from the other works: for some parameter α between $[0, 1]$, honest inputs are guaranteed to be aggregated at most once with at least α fraction of other inputs from honest users.

Table 2: Total received and sent asymptotic communication cost for all rounds per aggregation with semi-honest security. n denotes the total number of users, m is the size of the committee, k the security parameter, L the length of the input vector, and ℓ the bit length of each element.

Protocol	**Communication Cost**				
	Server		User		
	Setup	Agg.	Setup	Agg.	
BIK+17[17]	-	$O(nL\ell + n^2k)$	-	$O(L\ell + nk)$	
BBG+20[13]	-	$O(nL\ell + nk \log n)$	-	$O(L\ell + k \log n)$	
Flamingo[68]	$O(k \log n)$	$O(nL\ell + nk \log^2 n)$	$O(k \log n)$	$O(L\ell + nk \log n)$	$O(\mathsf{m}^2 + n \log n)$
LERNA [62]	-	$O(Lnk + \mathsf{m}^2 \cdot k)$	$O(\kappa^2 k)$	$O(\kappa L + L \log n + k)$	$O(L\kappa + L \log n + k)$
SASH [65]	-	$O(nL\ell + \kappa^2 k)$	-	$O(L\ell + nk)$	
OPA	-	$O(Ln + \mathsf{m})$	-	$O(kL + \mathsf{m})$	$O(n)$
OPA'	-	$O(kL(n + \mathsf{m}))$	-	$O(k(L + \mathsf{m}L))$	$O(k(nL + L))$

in n, our approach maintains a logarithmic complexity for the users which is noteworthy when considering our protocol's concurrent reduction in the number of communication rounds. Furthermore, our committee framework demonstrates a linear relationship with n for the committee members, a notable improvement compared to the linearithmic complexity and setup requirement in the case of [68] which considers a stateful set of decryptors (committee), as opposed to our stateless committee.[3]

When it comes to user communication and message sizes, previous solutions entail user complexities that either scale linearly [17, 65] or linearithmically [68] with the number of participants (n) according to Table 2. However, in our case, user communication complexity is reduced to a logarithmic level. Our communication among committee members exhibits a linear relationship with n. Furthermore, as the number of users n increases, the communication load placed on the server is also effectively reduced in comparison to other existing protocols. That said, the above advantages underline the scalability and efficiency of our protocols in the federated learning context which typically requires a very large number of n and L.

The works of [17, 13, 68] address an active adversary that can provide false information regarding which users are online on behalf of the server, by requiring users to verify the signatures of the claimed online set. This approach introduces an additional two rounds into each protocol, resulting in 10 rounds in [17, 13] and 5 rounds in [68] with 10 rounds of setup. The setup communication complexity of [68] also increases to $O(k \log^2 n)$. In our work, we solve this issue without signatures. Note that prior work often required a two-thirds honest majority for the malicious setting. One can leverage this to instead introduce a gap between the reconstruction threshold r and corruption threshold t. Specifically, if $\mathsf{r} > (\mathsf{m} + \mathsf{t})/2$, then a malicious server can only reconstruct with respect to a unique set and any other information is purely random information, unhelpful to the server.

Comparison with LERNA [62]. LERNA requires a fixed, stateful committee (like Flamingo) to secret share client keys, whereas we support smaller, dynamic stateless committees that can change in every round. Concretely, LERNA works by having *each* client (in the entire universe of clients, not just for that iteration) secret-share the keys with the committee. Consequently, LERNA's committee needs to be much larger (2^{14} members for $\kappa = 40$ due to the number of shares they receive) and tolerate fewer dropouts, compared to our approach. Furthermore, LERNA's benchmarks assume 20K+ clients, while real-world deployments have 50-5000 clients per iteration. When the client count is low, the committee has to do significantly more work to handle and store the required large number of shares. That said, LERNA is not suitable for traditional FL applications. In the table, we use the same notations, as for LERNA to refer to committee size by utilizing κ in the committee calculations. LERNA could work for less than 16K parties but then the computation of committee members increases significantly as the number of parties decreases. Even concrete costs require the client to send 2GB of data during the setup phase (with 20K clients and $L = 50,000$). Per iteration, the cost is 0.91 MB. For the same parameters, $\mathsf{OPA}_{\mathsf{CL}}$ (the construction based on distributed key-homomorphic

[3] Flamingo [68] employed *decryptors* which were a random subset of clients chosen by the server to interact with it to remove masks from masked data that were sent by the larger set of clients. [68] lacks security guarantees in the event of collusion among all decryptors.

PRF in CL framework), requires the client to send 5.6 MB (across both server and committee, per iteration). As a result, LERNA only becomes cost-effective after more than 400 iterations, during which it requires a fixed, stateful, and large committee to stay alive. Recall that $\mathsf{OPA}_{\mathtt{LWR}}$ is even more optimal, in terms of communication cost than $\mathsf{OPA}_{\mathsf{CL}}$. However, LERNA's use of Ring-LWR assumption could provide for improved computation, when compared to both $\mathsf{OPA}_{\mathsf{CL}}$ and $\mathsf{OPA}_{\mathtt{LWR}}$.

Comparison with Willow [14]. The concurrent work, Willow, employs a static, stateful committee of members, with non-committee members (clients) communicating only once, offering improvements over previous approaches like LERNA. While Willow supports smaller committee sizes and client-to-committee communication that remains independent of committee size—compared to LERNA—this advantage comes with trade-offs. Specifically, committee members must:

1. Undergo a one-round setup procedure at the start of each training iteration, as secret key shares are revealed to account for dropout committee members.

2. Participate in two rounds of communication during the decryption phase: (1) to threshold-decrypt the aggregated ciphertext, and (2) to provide secret key shares for dropout committee members.

In both Willow and LERNA, only non-committee members communicate only once; however, OPA differs substantially. In our approach, all client parties (including committee members) communicate just once, with no need for a setup procedure. Additionally, our stateless committee members can function as clients, joining or leaving the system at any time, making it more adaptable and closer to the standard federated learning model

Finally, unlike OPA, Willow requires another party (or a set of parties) called verifier(s) to ensure security against a malicious server. In essence, the verifier certifies that the server has not behaved maliciously. While we refer the reader to their paper for the details of the role of the verifier, it is sufficient to note that the role of the verifier is to verify information about the ciphertexts sent by each client to the committee members. This involves $\omega(n)$ performance of the verifier. However, by using a tree-based approach and increasing the number of verifiers, Willow guarantees that, while all the verifiers together perform $O(n)$ work, each verifier only does $o(n)$ work.

3 Related Work

Multi-Round Private Summation. We begin by revisiting the concept outlined in [54]. The first multi-round secure aggregation protocol, designed to enable a single server to learn the sum of inputs $x_1, \ldots, x_n$ while hiding each input x_i is based on the following idea. Each user i adds a mask r_i to their private input x_i. This mask remains hidden from both the server and all other users it exhibits the property of canceling out when combined with all the other masks, i.e., $\sum_{i\in[n]} r_i = 0$. Subsequently, each user forwards $X_i = x_i + r_i$ to the server. By aggregating all the X_i values, the server is then able to determine the sum of all x_i. More specifically, to generate these masks, a common key $k_{ij} = k_{ji} = \mathrm{PRG}(g^{s_i s_j})$ is established by every pair of clients i, j. Here, g^{s_i} serves as an ephemeral "public key" associated with each client $i \in [n]$. This public key is shared with all other clients during an initial round, facilitated through the server. Importantly, the value s_i remains secret by each client i. Then, each client $i \in [n]$ computes the mask $r_i = \sum_{j<i} k_{ij} - \sum_{j>i} k_{ij}$ and due to the cancellation property the server outputs $\sum_i X_i = \sum_i x_i$. In this protocol, users are required to engage in multiple rounds of communication, where each user communicates more than once. Moreover, the protocol does not permit users to drop out from an aggregation iteration.

Non-Interactive Private Summation with trusted setup. If we were to require users to communicate only once in a protocol iteration, we encounter the challenge of mitigating residual attacks. In a prior study conducted by [83], a solution based on DDH was proposed to mitigate residual attacks by involving a trusted setup that assumes the generation of the common keys $k_{ij} = k_{ji}$ into the protocol. However, it is important to highlight that this particular setup lacked the necessary mechanisms to accommodate dropouts and facilitate dynamic participation for multiple aggregation iterations. An additional limitation of this construction is the necessity of establishing a trusted setup that can be utilized across multiple iterations. Furthermore, to ensure that the server is unable to recover the masking key given a client's masked inputs, the work relies on the DDH Assumption. An unfortunate consequence is that the server has to compute the discrete logarithm to recover the aggregate, a computationally expensive operation, particularly when dealing with large exponents. Numerous other works within this framework have emerged, each relying on distinct assumptions, effectively sidestepping the requirement for laborious discrete logarithm calculations. These include works based on the DCR assumption [56, 15], and lattice-based cryptography [11, 47, 86, 85, 90, 73].

Multi-Round Private Summation without Trusted Setup. A separate line of research endeavors to eliminate the necessity for a trusted setup by introducing multi-round decentralized reusable setups designed to generate masks while adhering to the crucial cancellation property. However, akin to the previously mentioned approaches, these protocols come with a caveat—they do not accommodate scenarios involving dropouts or dynamic user participation across multiple iterations. Dipsauce [22] is the first to formally introduce a definition for a distributed setup PSA with a security model based on k-regular graph, non-interactive key exchange protocols, and a distributed randomness beacon [36, 46, 77] to build their distributed setup PSA. Meanwhile, the work of Nguyen *et al.* [72], assuming a PKI (or a bulletin board where all the public keys are listed), computed the required Diffie-Hellman keys on the fly to then build a one-time decentralized sum protocol which allowed the server to sum up the inputs *one-time*, with their construction relying on class group-based cryptography. To facilitate multiple iterations of such an aggregation, they combined their one-time decentralized sum protocol with Multi-client Functional Encryption (MCFE) to build a privacy-preservation summation protocol that can work over multiple rounds, without requiring a trusted setup and merely requiring a PKI. Unfortunately, per iteration, the clients need to be informed of the set of users participating in that round and unfortunately, they cannot drop out once chosen.

Non-Interactive Private Summation with a collector. To circumvent the need for a trusted setup and multi-round decentralized arrangements, an approach is presented in the work of [61] which introduces an additional server known as the "collector". The fundamental premise here is to ensure that the collector and the evaluation server do not collude, thus effectively mitigating the risks associated with residual attacks. This protocol does allow dynamic participation and dropouts per iteration.

Multi-Round Private Summation with Dynamic Participation (aka Secure Aggregation). Secure aggregation of users' private data with the aid of a server has been well-studied in the context of federated learning. Given the iterative nature of federated learning, dynamic participation is crucial. It enables seamless integration of new parties and those chosen to participate in various learning iterations, while also addressing the challenge of accommodating parties that may drop out during the aggregation phase due to communication failures or delays. Furthermore, an important problem in federated learning with user-constrained computation and wireless network resources is the computation and communication overhead which wastes bandwidth, increases training time, and can even impact the model accuracy if many users drop out. Seminal contributions by Bonawitz *et al.* [17] and Bell *et al.* [13] have successfully proposed secure aggregation protocols designed to cater to a large number of users while addressing the dropout challenge in a federated learning setting. However, it's important to note that these protocols come with a notable drawback—substantial round complexity and overhead are incurred during each training iteration. Even in the extensive corpus of research based on more complex cryptographic machinery (see [57] for a plethora of previous works) such as threshold additive homomorphic encryption etc., these persistent drawbacks continue to pose challenges. Notably, all follow up works [13, 62, 52, 68, 12, 65, 66] of [17] require multiple rounds of interaction based on distributed setups. Secure aggregation protocols, with their adaptable nature, hold relevance across a wide array of domains. They are applicable in various scenarios, including ensuring the security of voting processes, safeguarding privacy in browser telemetry as illustrated in [39], and facilitating data analytics for digital contact tracing, as seen in [6] besides enabling secure federated learning.

It is also important to note that some of these works - ACORN [12] and RoFL [66] build on top of the works of [17, 13] to tackle the problem of "input validation" using efficient zero-knowledge proof systems. The goal is for the clients to prove that the inputs being encrypted are "well-formed" to prevent poisoning attacks.RoFL allows for detection when a malicious client misbehaves, while ACORN presents a *non-constant* round protocol to identify and remove misbehaving clients. We leave it as a direction for future research on how to augment our protocol to also support input validation. The above discussion is also summarized in Table 3, by looking at four properties (a) whether the aggregate can be efficiently recovered, (b) whether it

Table 3: Comparison of Various Private Summation Protocols. TD stands for trusted dealer/trusted setup, DS stands for multi-round distributed setup. Note that DS implies several rounds of interaction while our protocol does not require any interaction. Here, efficient aggregate recovery refers to whether the server can recover the aggregate efficiently. For example, [83, 15, 52] require some restrictions on input sizes to recover the aggregate due to the discrete logarithm bottleneck.

	Efficient Aggregate	Dynamic Participation	TD vs DS	Assumptions
[83]	✗	✗	TD	DDH
[56]	✓	✗	TD	DCR
[15]	✗/✓	✗	TD	DDH/DCR
[11]	✓	✗	TD	LWE/R-LWE
[87]	✓	✗	TD	R-LWE
[90]	✓	✗	TD	AES
[85]	✓	✗	TD	RLWE
[61]	✗	✓	TD	DCR
[47]	✓	✗	TD	LWR
[22]	✗	✗	DS	LWR
[17, 13]	✓	✓	DS*	DDH
[52]	✓	✓	DS	DDH
[68]	✓	✓	DS	DDH
[62]	✓	✓	DS	DDH
Our Work	✓	✓	NA	HSM, LWR, (R)LWE

allows dynamic participation, (c) whether it requires trusted setup or multi-round distributed setup, and (d) the security assumptions.

3.1 OPA vs Other Communication/Computation Models

Shuffle Model. Note that our model bears similarities to the shuffle model, in which clients dispatch input encodings to a shuffler or a committee of servers responsible for securely shuffling before the data reaches the server for aggregation as in the recent work of Halevi et al. [53]. Nonetheless, it is important to note that such protocols typically entail multiple rounds among the committee servers to facilitate an efficient and secure shuffle protocol.

Multi-Server Secure Aggregation Protocols. It's worth emphasizing that multi-server protocols, as documented in [49, 40, 3, 78, 94], have progressed to a point where their potential standardization by the IETF, as mentioned in [75], is indeed noteworthy. In the multi-server scenario, parties can directly share their inputs securely among a set of servers, which then collaborate to achieve secure aggregation. Some of the works in this domain include two-server solutions Elsa [79] and SuperFL [94] or the generic multi-server solution Flag [10]. Unfortunately, in the case of federated learning, which involves handling exceptionally long inputs, the secret-sharing approach becomes impractical due to the increase in communication complexity associated with each input. Furthermore, these servers are required to have heavy computation power and be stateful (retaining data/state from iteration to iteration). Jumping ahead, in our protocol the ephemeral parties are neither stateful nor require heavy computation.

Commmittee-Based MPC. Committee-based MPC is widely used for handling scenarios involving a large number of parties. However, it faces a security vulnerability known as adaptive security, where an adversary can corrupt parties after committee selection. The YOSO model, introduced by Gentry *et al.* [?] proposes a model that offers adaptive security. In YOSO, committees speak once and are dynamically changed in each communication round, preventing adversaries from corrupting parties effectively. The key feature of YOSO is that the identity of the next committee is known only to its members, who communicate only once and then become obsolete to adversaries. YOSO runs generic secure computation calculations and aggregation can be one of them. However, its efficiency is prohibitive for secure aggregation. In particular, the communication complexity of YOSO in the computational setting scales quadratic with the number of parties n (or linear in n if the cost is amortized over n gates for large circuits). Additionally, to select the committees, an expensive role assignment protocol is applied. Like LERNA, in YOSO also specific sizes for the committee need to be fulfilled to run a protocol execution. Last but not least, our protocol does not rely on any secure role assignment protocol to choose the committees since even if all committee members are corrupted, privacy is still preserved. Fluid MPC [37, 16] also considers committee-based general secure computation. However, like YOSO, it is not practical. Unlike YOSO, it lacks support for adaptive security.

Moreover, SCALES [2] considers ephemeral servers a la YOSO responsible for generic MPC computations where the clients only provide their inputs. This approach is of theoretical interest as it is based on heavy machinery such as garbling and oblivious transfer if they were to be considered for the task of secure aggregation. Moreover, SCALES needs extra effort to hide the identities of the servers which we do not require.

4 Technical Overview

In this work, we focus on building a primitive, One-shot Private Aggregation (OPA), that enables privacy-preserving aggregation of multiple inputs, across several aggregation iterations whereby a client only speaks once on his will, per iteration.

Seed-Homomorphic PRG (SHPRG). A PRG $G : \mathcal{K} \to \mathcal{Y}$ is said to be seed-homomorphic if $G(s_1 \oplus s_2) = G(s_1) \otimes G(s_2)$ where $(\oplus, \mathcal{K})$ and $(\otimes, \mathcal{Y})$ are groups. We show how to build SHPRG from $\mathtt{LWR}$ and $\mathtt{LWE}$ Assumption. Note that the former was known but the latter was not. For ease of exposition, we will focus on the $\mathtt{LWR}$ construction. For a randomly chosen $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{L \times n}$, $\mathsf{PRG}_{\mathtt{LWR},\mathbf{A}}(\mathbf{s}) = \lfloor \mathbf{A}\mathbf{s} \rceil_p$ where $p < q$. Unfortunately, this construction is only almost seed-homomorphic in that there is an induced error. This is formally defined in Construction 1.

Secret Sharing over Finite Fields. In standard Shamir secret sharing [82], one picks a secret s and generate a polynomial $f(X) = \sum_{i=0}^{\mathsf{r}-1} a_i \cdot X^i$ where $a_0 = s$ and $a_1, \ldots, a_{\mathsf{r}-1}$ are randomly chosen from the field. Assuming there are m parties, the share for party $i \in [\mathsf{m}]$ is $f(i)$, and any subset of at least r parties can reconstruct s and any subset of $\mathsf{r} - 1$ shares are independently random. We can lower the corruption threshold from $\mathsf{r} - 1$ to t to obtain additional properties from the scheme. In packed secret sharing [48], one can hide multiple secrets using a single polynomial.

One-shot Private Aggregation (based on SHPRG). With the needed background in place, we are ready to build the new primitive called One-shot Private Aggregation. Critically, we want to empower a client to speak once, per iteration, and help the server successfully aggregate long vectors (of length L, say). To this end, we have the client communicate at the same time with the server and to a set of committee members, via the server. We will assume that there is a PKI or a mechanism to retrieve the public key of these committee members so that the communication to the committee can be encrypted. These are m ephemerally chosen clients who are tasked with helping the server aggregate, for that iteration. Flamingo [68, Figure 1, Lines 2-7] presents an approach on how to sample these committee members using a randomness beacon such as [46]. Observe that the one-shot communication can be leveraged to successfully avoid the complex setup procedures that were reminiscent of prior work. Instead, the client i samples a seed from the seed space of the PRG, i.e., $\mathsf{sd}_i \leftarrow_\$ \mathsf{PRF}.\mathcal{K}$. Then, the PRG is expanded under this seed sd_i and effectively serves as a mask for input $\mathbf{x}_{i,\ell}$. Here ℓ is the current iteration number. In other words, it computes $\mathbf{ct}_{i,\ell} = \mathbf{x}_{i,\ell} + \mathsf{PRG.Expand}(\mathsf{sd}_i)$. Intuitively, the PRF security implies that the evaluation is pseudorandom and is an effective mask for the input. Then, the client secret shares using standard Shamir's Secret Sharing, sd_i to get shares $\left\{\mathsf{sd}_i^{(j)}\right\}_{j\in[\mathsf{m}]}$. Share $\mathsf{sd}_i^{(j)}$ is sent to committee member j, via the server through an appropriate encryption algorithm. Each committee member simply adds up the shares, using the linear homomorphism property of Shamir's secret sharing. After receiving from the clients for that round, the committee member j sends the added shares, which corresponds to $\sum_{i=1}^{n} \mathsf{sd}_i^{(j)}$. The server can successfully reconstruct from the information sent by the committee to get $\sum_{i=1}^{n} \mathsf{sd}_i$. Note that adding up the ciphertext from the clients results in $\sum_{i=1}^{n} \mathbf{ct}_{i,\ell} = \sum_{i=1}^{n} \mathbf{x}_{i,\ell} + \sum_{i=1}^{n} \mathsf{PRG.Expand}(\mathsf{sd}_i)$. Key Homomorphism of the PRF implies that $\sum_{i=1}^{n} \mathsf{PRG.Expand}(\mathsf{sd}_i) = \mathsf{PRG.Expand}\left(\sum_{i=1}^{n} \mathsf{sd}_i\right)$. Note that the server, with the reconstructed information, can compute $\mathsf{PRG.Expand}\left(\sum_{i=1}^{n} \mathsf{sd}_i\right)$ and subtract it from $\sum_{i=1}^{n} \mathbf{ct}_{i,\ell}$ to recover the intended sum. This is the core idea behind our construction. However, there are a few caveats. First, the server recovers the sum of the keys $\sum_{i=1}^{n} \mathsf{sd}_i$, which constitutes a leakage on the seed of the PRG. In other words, we require a leakage-resilient, seed homomorphic PRG. Second, LWR and the LWE based construction of seed homomorphic PRG are only almost seed homomorphic. Thus, care needs to be taken to encode and decode to ensure that the correct sum is recovered. We show that the LWR and LWE based schemes are key-homomorphic and leakage-resilient while describing the necessary encoding and decoding algorithms for the input. A key benefit of this construction is that the key shared with the committee is usually independent of the vector length. Our proof of security is in the simulation-based paradigm against both a semi-honest and an active server.

One-shot Private Aggregation against Malicious Clients. Prior work to ensure the detection of client misbehavior such as ACORN [12] requires the usage of verifiable secret sharing. Unfortunately, this is an expensive process as the committee member is required to do at r exponentiations, per client. Instead, we take an alternative route by designing a publicly verifiable secret-sharing scheme. This employs SCRAPE [27] and combines it with a simple Σ-protocol. As a result, the server can verify that given the commitment to shares, the shares are indeed a valid Shamir Secret Sharing. Meanwhile, the committee member simply needs to verify if the commitment to the share matches its encrypted share with the committee member raising an alarm should the check fail. This is a much leaner verification process for the committee members.

CL Framework. We use the generalized version of the framework, as presented by Bouvier *et al.* [20]. Broadly, there exists a group $\widehat{\mathbb{G}}$ whose order is $\mathsf{M} \cdot \hat{s}$ where $\gcd(\mathsf{M}, \hat{s}) = 1$ and $\hat{s}$ is unknown while M is a parameter of the scheme. Then, $\widehat{\mathbb{G}}$ admits a cyclic group $\mathbb{F}$, generated by f whose order is M. Consider the cyclic subgroup $\mathbb{H}$ which is generated by $h = x^{\mathsf{M}}$, for a random $x \in \widehat{\mathbb{G}}$. Then, one can consider the cyclic subgroup $\mathbb{G}$ generated by $g = f \cdot h$ with $\mathbb{G}$ factoring as $\mathbb{F} \cdot \mathbb{H}$. The order of $\mathbb{G}$ is also unknown. The $\mathtt{HSM}_{\mathsf{M}}$ assumption states that an adversary cannot distinguish between an element in $\mathbb{G}$ and $\mathbb{H}$, while a discrete logarithm is easy in $\mathbb{F}$. Meanwhile, $\bar{s}$, an upper bound for $\hat{s}$ is provided as input. Note that for $\mathsf{M} = N$ where N is an RSA modulus, the $\mathtt{HSM}_{\mathsf{M}}$ assumption reduces to the DCR assumption. Therefore, the $\mathtt{HSM}_{\mathsf{M}}$ assumption can be viewed as a generalization of the DCR assumption.

Secret Sharing over Integers. Braun *et al.* [21] was the first to identify how to suitably modify Shamir's secret sharing protocol to ensure that the operations can work over a group of unknown order, such as the ones we use on the CL framework. This stems from two reasons. The first is leakage in that a share $f(i)$ corresponding to some sharing polynomial f always leaks information about the secret s, mod$i$ when the operation is over the set of integers. Meanwhile, the standard approach to reconstruct the polynomial requires the computation of the Lagrange coefficients which involves dividing by an integer, which again needs to be "reinterpreted" to work over the set of integers. The solution to remedy both these problems is to multiply with an offset $\Delta = \mathsf{m}!$ where m is the total number of shares.

(Almost) Key Homomorphic Pseudorandom Functions. The concept of key homomorphic PRFs (KH-PRFs) was introduced by [71], who demonstrated that $H(x)^k$ is a secure KH-PRF under the DDH assumption in the Random Oracle Model, where H is a hash function, k is the key, and x is the input. [19] later constructed an almost KH-PRF under the Learning with Rounding assumption [9], which was formally proven secure by [47]. Our work leverages almost KH-PRF constructions from both LWR and LWE assumptions in the Random Oracle Model. Additionally, we present a novel KH-PRF construction in the CL framework, yielding new constructions under the HSM assumption (including DCR-based constructions). We adapt the DDH-based construction to the CL framework and prove that $F(k, x) = \mathtt{H}(x)^k$, where $k \leftarrow_{\$} \mathcal{K}$ and $\mathtt{H} : \{0,1\}^* \to \mathbb{H}$ is modeled as a random oracle, is a secure KH-PRF under the HSM assumption. This adaptation requires careful consideration of appropriate groups, input spaces, and key spaces.

Distributed Key Homomorphic Pseudorandom Functions (KHPRF). [19] presented generic constructions of Distributed PRFs from any KH-PRF using secret sharing techniques. However, the CL framework's use of groups with unknown order necessitates working over integer spaces. While Linear Integer Secret Sharing [43] exists, it can be computationally expensive. Instead, we utilize Shamir Secret Sharing over Integer Space as described by [21], refining it with appropriate offsets to construct Distributed PRFs from our HSM-based construction. In other words, rather than reconstructing the secret, the solution reconstructs a deterministic function of the secret which is accounted for in the PRF evaluation. This induces complications in reducing the security of our distributed key homomorphic PRF to that of the HSM-based KH-PRF. Finally, we demonstrate that our construction maintains the key homomorphism property, allowing combinations of partial evaluations of secret key shares to match the evaluation of the sum of keys at the same input point.

Building Distributed Key Homomorphic PRFs. [19] introduced a generic construction of distributed KHPRFs from almost key homomorphic PRFs. They proposed $F_{\mathtt{LWR}}(\mathbf{k}, x) := \lfloor\langle \mathtt{H}(x), \mathbf{k}\rangle\rfloor_p$ as an almost key homomorphic PRF, where $q > p$ are primes, $\mathbf{k} \leftarrow_{\$} \mathbb{Z}_q^{\ell}$, and $\mathtt{H}$ is a suitably defined hash function. Their reduction to build a distributed PRF utilizes standard Shamir's Secret Sharing over fields, simplifying the process compared to integer secret sharing due to the prime nature of q and p. However, their proposed construction contained shortcomings affecting both correctness and security proofs. We will now provide a brief overview of these issues. An almost KH-PRF satisfies $F(k_1 + k_2, x) = F(k_1, x) + F(k_2, x) - e$ for some error e. For the $\mathtt{LWR}$ construction, $e \in 0, 1$. However, this implies $F(T \cdot k, x) = T \cdot F(k, x) - e_T$ where $e_T \in 0, \ldots, T-1$ for any integer T, leading to error growth and affecting Lagrange interpolation. The authors proposed multiplying by an offset $\Delta = \mathsf{m}!$ to bound the error and use rounding to mitigate its impact by ensuring that the error terms are "eaten" up. However, their security reduction faces challenges when simulating partial evaluations for unknown i^*. Lagrange interpolation with the "clearing out the denominator" technique causes further error growth, necessitating additional rounding. Consequently, the challenger can only provide $\left\lfloor \Delta F(k^{(i)}, x)\right\rfloor_u$. Thus, their definition of partial evaluation function needs to be updated to be consistent with what is simulatable. We identified further issues in their rounding choices. Specifically, partial evaluations should be rounded down to u where $\lfloor p/u\rfloor > (\Delta + 1) \cdot \mathsf{r} \cdot \Delta$ (r being the reconstruction threshold). Moreover, their framework only addressed single-key PRFs, not vector-key cases like LWR. We address these issues in our construction of a distributed, almost key homomorphic PRF based on LWR. We formally prove that $F_{\mathtt{LWR}}(\mathbf{k}, x) = \left\lfloor \Delta \left\lfloor \Delta \left\lfloor\langle \mathtt{H}(x), \mathbf{k}\rangle_p\right\rfloor_u\right\rfloor\right\rfloor_v$ for appropriate choices of u and v is a secure, distributed, almost key homomorphic, PRF.

One-shot Private Aggregation without Leakage Simulation. The construction of OPA' is similar to the earlier ones based on seed-homomorphic PRG. There exist the following differences:

- A client i has to sample L different keys. Each key in is used to evaluate the PRF at a point ℓ and is used to mask the input $x_i^{(in)}$.
- It then secret shares each of the L keys by running the $\mathsf{DPRF.Share}$, the algorithm to generate the shares of the DPRF key.
- Each share for committee member j is evaluated at ℓ. This evaluation is sent to the committee member.
- Finally, the server runs $\mathsf{DPRF.Combine}$ to combine the information from the committee members to get the PRF evaluation at ℓ under the sum of the keys, for each index in. $\mathsf{Combine}$ is the algorithm that helps reconstruct the evaluation from partial evaluations.

Stronger Security Definitions and Construction. We also present a stronger security guarantee, which was not provided by the committees of Lerna and Flamingo, whereby the committee members can all collude and observe all encrypted ciphertexts and all auxiliary information, and cannot mount an IND-CPA-style attack. Unfortunately, our current construction where the inputs are solely blinded by the PRF evaluation, which is also provided to the committee members in shares, can be unblinded by the committee leaking information about the inputs. We modify the syntax where each label/iteration begins with the server, which has its secret key, advertising a "public key" for that iteration (the keys for all iterations can also be published beforehand). The auxiliary information sent by a client to the committee is a function of this public key while the actual ciphertext is independent of this public key. Intuitively, this guarantees that the committee member's information is "blinded" by the secret key of the server and cannot be used to unmask the information sent by the client. We now describe our updated construction. For each label ℓ, the server publishes $F(\mathsf{k}_0, \ell)$ where k_0 is its public key. Then, the auxiliary information sent by the client is of the form $F(\mathsf{k}_i^{(j)}, \ell)$ where $\mathsf{k}_i^{(j)}$ is the j-th share of the i-th clients key. Client i masks its input by doing $f^{x_i} \cdot F(\mathsf{k}_0, \ell)^{\mathsf{k}_i}$. Committee member j combines the results to then send $F(\sum_{i=1}^n \mathsf{k}_i^{(j)}, \ell)$ to the server. The server uses Lagrange interpolation and its own key k_0 to compute: $F(\mathsf{k}_0 \sum_{i=1}^n \mathsf{k}_i, \ell)$. Meanwhile, the server, upon multiplying the ciphertexts gets $X_\ell = f^{\sum_{i=1}^n x_{i,\ell}} \cdot F(\mathsf{k}_0 \sum_{i=1}^n \mathsf{k}_i, \ell)$. The recovery is straightforward after this point.

5 Preliminaries and Cryptographic Building Blocks

Notations. For a distribution X, we use $x \leftarrow\$ X$ to denote that x is a random sample drawn from the distribution X. We denote by $\mathbf{u}$ a vector and by $\mathbf{A}$ a matrix. For a set S we use $x \leftarrow\$ S$ to denote that x is chosen uniformly at random from the set S. By $[n]$ for some integer n, we denote the set $\{1, \ldots, n\}$.

Cryptographic Preliminaries. This is deferred to Section A. In Section A.1, we discuss the definition and construction of secret-sharing schemes over a field and over integers. In Section A.2, we present lattice-based assumptions. This is followed by an exposition on the syntax and security definitions of various types of pseudorandom functions through Sections A.3 and A.4. Finally, we introduce the CL Framework in Section A.5.

Now, we will present the syntax and security of a seed-homomorphic PRG, while also presenting constructions from lattice-based assumptions.

5.1 Seed Homomorphic PRG

5.1.1 Syntax and Security

Definition 1 (Seed Homomorphic PRG (SHPRG))**.** *Consider an efficiently computable function* $\mathsf{PRG} : \mathcal{K} \to \mathcal{Y}$ *where* $(\mathcal{K}, \oplus), (\mathcal{Y}, \otimes)$ *are groups. Then* $(\mathsf{PRG}, \oplus, \otimes)$ *is said to be a secure seed homomorphic pseudorandom generator (SHPRG) if:*

- PRG *is a secure pseudorandom generator (PRG), i.e., for all PPT adversaries* $\mathcal{A}$*, there exists a negligible function* negl *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{PRG}} \leftarrow\$ \mathsf{PRG.Gen}, b \leftarrow\$ \{0,1\}, \mathsf{sd} \leftarrow\$ \mathcal{K} \\ Y_0 = \mathsf{PRG.Expand}(\mathsf{sd}), Y_1 \leftarrow\$ cY \\ b' \leftarrow\$ \mathcal{A}(Y_b) \end{array} \right] \leqslant \frac{1}{2} + \mathrm{negl}(\kappa)$$

- *For every* $\mathsf{sd}_1, \mathsf{sd}_2 \in \mathcal{K}$*, we have that* $\mathsf{PRG.Expand}(\mathsf{sd}_1) \otimes \mathsf{PRG.Expand}(\mathsf{sd}_2) = \mathsf{PRG.Expand}(\mathsf{sd}_1 \oplus \mathsf{sd}_2)$

In the construction below, we abuse notation and simply write $\mathsf{PRG}(\mathsf{sd})$ as a shorthand for $\mathsf{PRG.Expand}(\mathsf{sd})$

5.1.2 Construction from LWR Assumption

Construction 1 (SHPRG from LWR Assumption)**.** Let $\mathbf{A} \leftarrow\$ \mathbb{Z}_q^{n_1 \times L}$ be the output of $\mathsf{PRG.Gen}$, then consider the following seed homomorphic PRG $\mathsf{PRG}_{\mathtt{LWR}} : \{0,1\}^{n_1} \to \{0,1\}^L$ where $L > n_1$ is defined as $\mathsf{PRG}_{\mathtt{LWR}, \mathbf{A}}(\mathsf{sd} = \mathbf{s}) = \lceil \mathbf{A}^\top \cdot \mathbf{s} \rfloor_p$ where $q > p$ with $\lceil x \rfloor_p = \lceil x \cdot p/q \rfloor$ for $x \in \mathbb{Z}_q$.

This is almost seed homomorphic in that: $\mathsf{PRG}(\mathbf{s}_1 + \mathbf{s}_2) = \mathsf{PRG}(\mathbf{s}_1) + \mathsf{PRG}(\mathbf{s}_2) + e$ where $e \in \{-1, 0, 1\}^L$

Theorem 1 (Leakage Resilience of Construction 1)**.** *Let* PRG_{LWR} *be the PRG defined in Construction 1. Then, it is leakage resilient in the following sense:*

$$\left\{\mathsf{PRG}_{LWR}(\mathsf{sd}) \bmod p, \mathsf{sd} + \mathsf{sd}' \bmod q : \mathsf{sd}, \mathsf{sd}' \leftarrow\$ \mathbb{Z}_q^{n_1}\right\} \approx_c \left\{\mathbf{y}, \mathsf{sd} + \mathsf{sd}' \bmod q : \mathbf{y} \leftarrow\$ \mathbb{Z}_p^L, \mathsf{sd}, \mathsf{sd}' \leftarrow\$ \mathbb{Z}_q^{n_1}\right\}$$

Proof. The proof proceeds through a sequence of hybrids.

$\mathsf{Hybrid}_0(\kappa)$**:** The left distribution is provided to the adversary. In other words, the adversary gets:

$$\left\{\mathsf{PRG}_{\mathtt{LWR}}(\mathsf{sd}) \bmod p, \mathsf{sd} + \mathsf{sd}' \bmod q : \mathsf{sd}, \mathsf{sd}' \leftarrow\$ \mathbb{Z}_q^{n_1}\right\}$$

$\mathsf{Hybrid}_1(\kappa)$**:** In this hybrid, we replace $\mathsf{sd} + \mathsf{sd}' \bmod q$ with a uniformly random value $\mathsf{sd}'' \leftarrow\!\!\$\ \mathbb{Z}_q^{n_1}$.

$$\left\{\mathsf{PRG}_{\mathtt{LWR}}(\mathsf{sd}) \bmod p, \mathsf{sd}'' : \mathsf{sd}, \mathsf{sd}'' \leftarrow\!\!\$\ \mathbb{Z}_q^{n_1}\right\}$$

Note that $(\mathsf{sd}+\mathsf{sd}') \bmod q$ and sd'' are identically distributed. Let us assume that there exists a leakage function oracle L that can be queried with an input sd, for which it either outputs $\mathsf{sd} + \mathsf{sd}' \bmod q$ for a randomly sampled $\mathsf{sd}' \leftarrow\!\!\$\ \mathbb{Z}_q^{n_1}$ or outputs $\mathbf{s}' \leftarrow\!\!\$\ \mathbb{Z}_q^{n_1}$. If one could distinguish between hybrids $\mathsf{Hybrid}_0, \mathsf{Hybrid}_1$, then one could distinguish between the outputs of the leakage oracle, but the outputs are identically distributed. Therefore, the $\mathsf{Hybrid}_0, \mathsf{Hybrid}_1$ are identically distributed. Therefore, the $\mathsf{Hybrid}_0, \mathsf{Hybrid}_1$ are identically distributed.

$\mathsf{Hybrid}_2(\kappa)$**:** In this hybrid, we will replace the PRG computation with a random value from the range.

$$\left\{\mathbf{y}, \mathsf{sd}'' : \mathbf{y} \leftarrow\!\!\$\ \mathbb{Z}_p^{L}, \mathsf{sd}'' \leftarrow\!\!\$\ \mathbb{Z}_q^{n_1}\right\}$$

Under the security of the PRG, we get that $\mathsf{Hybrid}_1, \mathsf{Hybrid}_2$ are computationally indistinguishable.

$\mathsf{Hybrid}_3(\kappa)$**:** We replace sd'' with $(\mathsf{sd} + \mathsf{sd}') \bmod q$.

$$\left\{\mathbf{y}, (\mathsf{sd} + \mathsf{sd}') \bmod q : \mathbf{y} \leftarrow\!\!\$\ \mathbb{Z}_p^{L}, \mathsf{sd}, \mathsf{sd}'' \leftarrow\!\!\$\ \mathbb{Z}_q^{n_1}\right\}$$

As argued before $\mathsf{Hybrid}_2, \mathsf{Hybrid}_3$ are identically distributed.

Note that Hybrid_3 is the right distribution from the theorem statement. This completes the proof. □

5.1.3 Construction from LWE Assumption

Construction 2 (SHPRG from LWE Assumption)**.** Let $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{L\times\lambda}$ be the output of $\mathsf{PRG.Gen}$. Then, consider the following seed homomorphic PRG $\mathsf{PRG}_{\mathtt{LWE}} : \mathbb{Z}_q^{\lambda} \times \chi^{L} \to \mathbb{Z}_q^{L}$ is defined as $\mathsf{PRG}_{\mathtt{LWE},\mathbf{A}}(\mathbf{s}, \mathbf{e}) = \mathbf{A}\mathbf{s} + \mathbf{e}$

The proof of security is a direct application of the LWE Assumption. However, we now prove the (almost) seed homomorphic property.

$$\begin{aligned}
\mathsf{PRG}_{\mathtt{LWE},\mathbf{A}}(\mathbf{s}_1, \mathbf{e}_1) &= \mathbf{A}\mathbf{s}_1 + \mathbf{e}_1\\
\mathsf{PRG}_{\mathtt{LWE},\mathbf{A}}(\mathbf{s}_2, \mathbf{e}_2) &= \mathbf{A}\mathbf{s}_2 + \mathbf{e}_2\\
\mathsf{PRG}_{\mathtt{LWE},\mathbf{A}}(\mathbf{s}_1, \mathbf{e}_1) + \mathsf{PRG}_{\mathtt{LWE},\mathbf{A}}(\mathbf{s}_2, \mathbf{e}_2) &= \mathbf{A}(\mathbf{s}_1 + s_2) + (\mathbf{e}_1 + \mathbf{e}_2) = \mathbf{A}(\mathbf{s}_1 + s_2) + \mathbf{e}
\end{aligned}$$

where $\mathbf{e} \leqslant \mathbf{e}_1 + \mathbf{e}_2 + 1$.

Looking ahead, when we use this PRG to mask the inputs, we will do the following: $\mathsf{PRG}_{\mathtt{LWE},\mathbf{A}}(s_i, e_i) = \mathbf{A}\mathbf{s}_i + \mathbf{e}_i + \lfloor q/p \rfloor \cdot \mathbf{x}_i$ where $\mathbf{x}_i \in \mathbb{Z}_p^m$. Upon adding n such ciphertexts $(\bmod q)$, we get:

$$\mathbf{A}\sum_{i=1}^{n} \mathbf{s}_i + \mathbf{e} + \lfloor q/p \rfloor \cdot \sum_{i=1}^{n} \mathbf{x}_i$$

where $\mathbf{e} \leqslant 1 + \sum_{i=1}^{n} \mathbf{e}_i$. Therefore, to eventually recover $\sum_{i=1}^{n} \mathbf{x}_i \bmod p$ from just the value of $\sum_{i=1}^{n} \mathbf{s}_i$, we will require $||\mathbf{e}||_\infty < \frac{q}{2p}$. Looking ahead, we will rely on the Hint-LWE Assumption 6 to show that it is leakage resilient when we use it to build our aggregation tool.

Remark 1 (Construction based on Ring-LWE)**.** One can also extend the above LWE construction to the Ring-LWE [67] setting.

Recall the R-LWE Assumption. Let N be a power of two, and let $m > 0$ be an integer. Let R be a cyclotomic ring of degree N, and let R_q be its residue ring modulo $q > 0$. Then, the following holds:

$$\left\{(\mathbf{a}, \mathbf{a}\cdot k + \mathbf{e}) : \mathbf{a} \leftarrow\!\!\$\ R_q^m, k \leftarrow\!\!\$\ R_q, \mathbf{e} \leftarrow\!\!\$\ \chi^m\right\} \approx_c \left\{(\mathbf{a}, \mathbf{u}) : \mathbf{a} \leftarrow\!\!\$\ R_q^m, \mathbf{u} \leftarrow\!\!\$\ R_q^m\right\}$$

This gives us the following construction: $\mathsf{PRG}_{\mathtt{R\text{-}LWE}}((k, e)) : \mathbf{a}k + \mathbf{e}$

6 One-shot Private Aggregation

In this section, we begin by introducing the syntax and security of our primitive which we call One-shot Private Aggregation (OPA). Broadly speaking, the goal of this primitive is to support a server (aka aggregator) to sum up the clients' inputs encrypted to a particular label (which can be a tag, timestamp, etc.), without it learning any information about the inputs beyond just the sum.

6.1 Simulation-Based Privacy

Our proof approach is based on the standard simulation-based framework [51, 63] where we demonstrate that any attacker against our protocol can be simulated by an attacker Sim in an ideal world where a trusted party $\mathcal{T}$ computes a function F on the clients' inputs X. In our case, this function is that of vector summation. We consider an attacker $\mathcal{A}$, that controls at most $\eta \cdot n$ clients and possibly the server. In our construction, we will consider these $\eta \cdot n$ clients to possibly include those belonging to the committee. The ideal world consists of the following steps which are adapted to the simpler setting where only one party, i.e., the server has the output:

(a) The honest clients provide the inputs to the trusted party $\mathcal{T}$.

(b) Sim chooses which corrupted clients send the input and which ones abort.

(c) If the server is corrupted, then Sim can either choose to abort the protocol or continue.

(d) If the protocol is not aborted, then $\mathcal{T}$ computes the function $F(X)$ and sends to the server.

(e) Finally, if the server is not corrupted, it outputs what it has received from $\mathcal{T}$.

Our function F is parametrized by the following: (a) the set of inputs $X = \{\mathbf{x}_{i,\ell}\}_{i \in [n]}$, (b) the set of client dropouts $D \subseteq [n]$, and (c) $\delta \in [0, 1)$ which is the maximum fraction of dropouts permitted. In addition, we also parametrize it by the iteration ID ℓ. Then,

$$F^{\ell}_{D,\delta} = \begin{cases} \sum_{i \in [n] \setminus D} \mathbf{x}_{i,\ell} & \textbf{if } |D| \leqslant \delta n \\ \perp & \textbf{otherwise} \end{cases} \tag{1}$$

6.2 Our Construction of One-shot Private Aggregation Scheme

The key idea behind our construction is that a seed-homomorphic PRG allows us to have a single seed, much shorter than L, to mask all the L inputs. Then, one can simply secret-share the seed, which reduces computation and communication. Our seed-homomorphic PRGs are in the standard model. However, it is important to note that for the intended application of Federated Learning one might have to rely on the random oracle model to thwart attacks such as those pointed out by Pasquini *et al.* [74]. Though the underpinning idea of OPA is the combination of seed-homomorphic PRG and an appropriate secret-sharing scheme, there are technical issues with presenting a generic construction. We begin by presenting the construction based on the Learning with Rounding Assumption.

6.2.1 Construction of $\mathsf{OPA}_{\mathtt{LWR}}$

We now present $\mathsf{OPA}_{\mathtt{LWR}}$ and prove its correctness and security. We rely on the seed-homomorphic PRG (Construction 1) and combine it with Shamir's Secret Sharing Scheme over $\mathbb{F}_q$ (Construction 6).

Construction 3. We present our construction in Figure 2.

Correctness. First, recall that Construction 1 is only almost seed homomorphic. In other words,

$$\mathsf{PRG}(\mathsf{sd}_1 + \mathsf{sd}_2) = \mathsf{PRG}(\mathsf{sd}_1) + \mathsf{PRG}(\mathsf{sd}_2) + e$$

where $e \in \{0, 1\}$.

For ease of presentation, our correctness proof is for $L = 1$, but it extends to any arbitrary L. Therefore, while the correctness of the Shamir's Secret Sharing scheme guarantees that sd_ℓ, computed in S-Combine, is indeed $\sum_{i=1}^{n} \mathsf{sd}_i \bmod q$, there is an error growth in AUX_ℓ. Specifically, we get that:

$$\mathsf{AUX}_\ell := \mathsf{PRG.Expand}\left(\mathsf{sd}_\ell = \sum_{i=1}^{n} \mathsf{sd}_{i,\ell}\right) = \sum_{i=1}^{n} \mathsf{PRG.Expand}(\mathsf{sd}_{i,\ell}) + e'$$

where $e' \in \{0, \dots, n-1\}$. We know that $\mathsf{Encode}(x_{i,\ell}) := n \cdot x_{i,\ell} + 1$.

$$\sum_{i=1}^{n} \mathsf{ct}_{i,\ell} - \mathsf{AUX}_\ell = \left(\sum_{i=1}^{n} (x_{i,\ell} \cdot n + 1) + \mathsf{Expand}\,(\mathsf{sd}_{i,\ell})\right) - \mathsf{Expand}\left(\sum_{i=1}^{n} \mathsf{sd}_{i,\ell},\right) \bmod p$$

Protocol Construction of $\mathsf{OPA}_{\mathtt{LWR}}$

One-Time System Parameters Generation

Run $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{SS.Setup}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{r}, 1^\mathsf{m})$
Run $\mathsf{pp}_{\mathsf{PRG}} \leftarrow \mathsf{PRG.Gen}(1^{\kappa_c}, 1^{\kappa_s})$
Set $\mathsf{pp} = (\mathsf{pp}_{\mathsf{PRG}}, \mathsf{t}, \mathsf{r}, \mathsf{m})$
return Committee of size m and pp.

Data Encryption Phase by Client i in iteration ℓ

Sample $\mathsf{sd}_{i,\ell} \leftarrow\!\!\$\ \mathsf{PRG}.\mathcal{K}$, $\mathsf{dig}_{i,\ell} \leftarrow\!\!\$\ \{0,1\}^{\log q}$
Compute $(x_1, \ldots, x_L) \leftarrow \mathsf{Encode}(\mathbf{x}_i)$
Compute $\mathbf{mask}_{i,\ell} = \mathsf{PRG.Expand}(\mathsf{sd}_{i,\ell})$, $\mathbf{mask}'_{i,\ell} := \mathtt{H}(\mathsf{dig}_{i,\ell})$
Compute $\mathbf{ct}_{i,\ell} = (x_1, \ldots x_L) + \mathbf{mask}_{i,\ell}$ $+\mathbf{mask}'_{i,\ell}$
$(\mathsf{sd}_i^{(j)})_{j\in[\mathsf{m}]} \leftarrow\!\!\$\ \mathsf{SS.Share}(\mathsf{sd}_{i,\ell}, \mathsf{t}, \mathsf{r}, \mathsf{m})$
$(\mathsf{dig}_{i,\ell}^{(j)})_{j\in[\mathsf{m}]} \leftarrow\!\!\$\ \mathsf{SS.Share}(\mathsf{dig}_{i,\ell}, \mathsf{t}, \mathsf{r}, \mathsf{m})$
for $j = 1, \ldots, \mathsf{m}$ **do**
  $\mathsf{aux}_{i,\ell}^{(j)} \leftarrow \mathsf{sd}_i^{(j)}, \mathsf{dig}_{i,\ell}^{(j)}$
Send $\mathbf{ct}_{i,\ell}$ to the Server
Send $c_{i,\ell}^{(j)} = \mathsf{Enc}_{\mathsf{pk}_j}(\ell, i, \mathsf{aux}_{i,\ell}^{(j)})$ to committee member j for each $j \in [\mathsf{m}]$, via server.

Set Intersection Phase by Server in iteration ℓ

For $j \in [\mathsf{m}]$, let $\mathcal{C}^{(j)} := \left\{i : c_{i,\ell}^{(j)} \text{was received by server}\right\}$
Let $C^{(0)} := \{i : \mathbf{ct}_{i,\ell} \text{was received by the server}\}$
Compute $\mathcal{C} := \cap_{j\in\mathcal{S}\cup\{0\}}\mathcal{C}^{(j)}$ // This is bit-wise AND operation of the $\mathsf{r}+1$ bit strings.
assert $|\mathcal{C}| \geqslant (1-\delta)n$
Send $\mathcal{C}, \left\{c_{i,\ell}^{(j)}\right\}_{i\in\mathcal{C}}$ for every j in $[\mathsf{m}]$

Data Combination Phase by Committee Member j in iteration ℓ

Recover $(i, \ell, \left\{\mathsf{aux}_{i,\ell}^{(j)} = (\mathsf{sd}_{i,\ell}^{(j)}, \mathsf{dig}_{i,\ell}^{(j)})\right\})$ from $\left\{c_{i,\ell}^{(j)}\right\}_{i\in\mathcal{C}}$
Verify $i \in \mathcal{C}$ and ℓ is the current iteration, else abort.
Compute $\mathsf{AUX}^{(j)} \leftarrow \sum_{i\in\mathcal{C}} \mathsf{sd}_{i,\ell}^{(j)}$
Send $\mathsf{AUX}^{(j)}$, $\left\{\mathsf{dig}_{i,\ell}^{(j)}\right\}_{i\in\mathcal{C}}$ to server

Data Aggregation Phase by Server in iteration ℓ

Let $\left\{\mathsf{AUX}_\ell^{(j)}\right\}_{j\in\mathcal{S}}, \{\mathbf{ct}_{i,\ell}\}_{i\in\mathcal{C}}$ be the inputs received by the server with $|\mathcal{S}| \geqslant \mathsf{r}$.
Run $\mathsf{sd}_\ell \leftarrow \mathsf{SS.Reconstruct}(\left\{\mathsf{AUX}_\ell^{(j)}\right\}_{j\in\mathcal{S}})$
$\mathsf{AUX}_\ell = \mathsf{PRG.Expand}(\mathsf{sd}_\ell)$
for $i \in \mathcal{C}$ **do**
  $\mathsf{dig}_{i,\ell} \leftarrow \mathsf{SS.Reconstruct}(\left\{\mathsf{dig}_{i,\ell}^{(j)}\right\}_{j\in\mathcal{S}})$
Compute $\mathbf{CT}_\ell \leftarrow \sum_{i\in\mathcal{C}} \mathbf{ct}_{i,\ell}$
Compute $(X_1, \ldots, X_L) = \mathbf{CT}_\ell - \mathsf{AUX}_\ell - \sum_{i\in\mathcal{C}} \mathtt{H}(\mathsf{dig}_{i,\ell})$
Compute $\mathbf{X}_\ell \leftarrow \mathsf{Decode}(\mathsf{pp}, (X_1, \ldots, X_L))$
return $\mathbf{X}_\ell$

Figure 2: Our Construction of OPA built from LWR-based Seed Homomorphic PRG (PRG) and Shamir's secret sharing scheme SS. Here, $\mathsf{Encode}(\mathbf{x}_{i,\ell}) := n \cdot \mathbf{x}_{i,\ell} + 1$ and $\mathsf{Decode}(X_i) := \lceil X_i/n \rceil - 1$. The lines are for security against an active server. The use of the second mask $\mathbf{mask}'_{i,\ell}$, as the output of a $\mathtt{H}$ is for simulation proof, for an active server. We will model $\mathtt{H}$ as a programmable random oracle.

$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n + \sum_{i=1}^{n} \mathsf{Expand}(\mathsf{sd}_{i,\ell}) - \mathsf{Expand}\left(\sum_{i=1}^{n} \mathsf{sd}_{i,\ell}\right) \bmod p$$

$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n + \mathsf{Expand}\left(\sum_{i=1}^{n} \mathsf{sd}_{i,\ell}\right) - \mathsf{Expand}\left(\sum_{i=1}^{n} \mathsf{sd}_{i,\ell}\right) - e' \bmod p$$

$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e' \bmod p$$

$$\bar{X}_\ell = n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e'$$

To make the last jump in the proof, we require:

$$0 \leqslant n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e' < p$$

First, $e' \leqslant n-1$. This guarantees that: $0 \leqslant n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e'$. Now, if $\sum_{i=1}^{n} x_{i,\ell} < (p-n)/n$ then we also get:

$$n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e' < p$$

Now, we show the correctness of Decode algorithm to recover $\sum_{i=1}^{n} x_{i,\ell}$ from $\bar{X}_\ell$.

- $\bar{X}_\ell/n = \sum x_{i,\ell} + (n-e')/n$
- $0 \leqslant e' \leqslant n-1 \Rightarrow 1/n \leqslant (n-e')/n \leqslant 1$
- Therefore, $\lceil \bar{X}_\ell/n \rceil = \sum x_{i,\ell} + 1$

Theorem 2. *Let κ_s and κ_c be the statistical and computational security parameters. Let L be the input dimension and n be the number of clients, that are $\mathrm{poly}(\kappa_c)$. Let δ be the dropout threshold and η be the corruption threshold such that $\delta + \eta < 1$. Then, there exists an efficient simulator Sim such that for all $\mathsf{K} \subset [n]$ such that $|\mathsf{K}| \leqslant \eta n$, inputs $X = \{\mathbf{x}_{i,\ell}\}_{i \in n \setminus \mathsf{K}}$, and for all adversaries $\mathcal{A}$ against Construction 3, that controls the server and the set of corrupted clients K, which behave semi-honestly (resp. maliciously), the output of Sim is computationally indistinguishable from the joint view of the server and the corrupted clients. Sim is allowed to query $F^{\ell}_{D,\delta}(X)$ (defined in Equation 1) once, per iteration.*

Proof is deferred to Section F.

Remark 2 (Malicious Server and Inconsistent Updates)**.** Recent work by Pasquini *et al.* [74] describes an attack where the server can send different models to different client updates with the goal that the model sent to a particular client can negate the training done by other clients on different models. In our case, this attack can be easily remedied with no overhead. Rather than evaluating the PRG with just the public matrix $\mathbf{A}$, one can first compute a hash $\mathtt{H}(\textit{model})$, and multiply it with $\mathbf{A}$. This would ensure that the matrix used by the client is tied to the model update sent. If different clients use different $\mathbf{A}$, the seed homomorphism fails. This would make it difficult, for a malicious server to send different models to different clients, to ensure that a particular client's contributions are not aggregated and therefore can be recovered. We can also simply switch to the key-homomorphic PRF constructions described in Section E.

In Section G, we model security when the entire committee is corrupted and the committee, through some attack, recovers the ciphertext of an honest client. The goal is to ensure that the input of the honest client is still preserved. In this definition, the server is honest.

6.2.2 Construction of $\mathsf{OPA}_{\mathtt{LWE}}$

Construction 4. As alluded to before, $\mathsf{OPA}_{\mathtt{LWE}}$ is largely similar to $\mathsf{OPA}_{\mathtt{LWR}}$ with the following differences:

- While the seed of $\mathsf{PRG}_{\mathtt{LWE}}$ is $(\mathsf{sd}, \mathbf{e})$, we will only secret share sd. We will argue below that the correctness still holds, for suitable definition of χ.
- The plaintext space for $\mathsf{OPA}_{\mathtt{LWE}}$, like the one for $\mathsf{OPA}_{\mathtt{LWR}}$, is $\mathbb{Z}_p$. Meanwhile the seed space for both $\mathsf{OPA}_{\mathtt{LWE}}$ and $\mathsf{OPA}_{\mathtt{LWR}}$ will be $\mathbb{Z}_q$. Let $\Delta := \lfloor q/p \rfloor$.
- We will use Shamir's Secret Sharing over q, as before, which is the seed space.
- There is a change in $\mathsf{S\text{-}Combine}$. To compute AUX_ℓ, the server uses the reconstructed seed sd_ℓ, and additionally sets the error component of the PRG seed to be 0.
- $\mathsf{Encode}(\mathbf{x}_{i,\ell}) := \Delta \cdot \mathbf{x}_{i,\ell}$
- $\mathsf{Decode}(X_i) := \lceil X_i/\Delta \rceil - 1$

Due to the similarities, we do not present the construction in its entirety. Meanwhile, we present the proof of correctness and proof of security.

Correctness. First, observe that Construction 1 is only almost seed-homomorphic, i.e.

$$\mathsf{PRG}((\mathsf{sd}_1 + \mathsf{sd}_2, \mathbf{e}), \ell) = \mathsf{PRG}((\mathsf{sd}_1, \mathbf{e}_1), \ell) + \mathsf{PRG}((\mathsf{sd}_2, \mathbf{e}_2), \ell) + \mathbf{e}'$$

for some error $\mathbf{e}'$. Indeed, assuming the correctness of Shamir's Secret Sharing, we get that the server computes:

$$\mathsf{AUX}_\ell := \mathsf{PRG.Expand}\left(\left(\sum_{i=1}^{n} \mathsf{sd}_i, 0\right), \ell\right) := \mathbf{A} \cdot \sum_{i=1}^{n} \mathsf{sd}_i$$

Meanwhile,

$$\begin{aligned}
\sum_{i=1}^{n} \mathsf{ct}_{i,\ell} &= \sum_{i=1}^{n} (\mathbf{A}\mathsf{sd}_i + \mathbf{e}_i + \Delta \cdot x_{i,\ell}) \\
&= \mathbf{A} \sum_{i=1}^{n} \mathsf{sd}_i + \sum_{i=1}^{n} \mathbf{e}_i + \sum_{i=1}^{n} \Delta \cdot x_{i,\ell} \\
\text{Let } X_\ell := \sum_{i=1}^{n} \mathsf{ct}_{i,\ell} - \mathsf{AUX}_\ell &= \sum_{i=1}^{n} \mathbf{e}_i + \sum_{i=1}^{n} \Delta \cdot x_{i,\ell} \\
\frac{X_\ell}{\Delta} &= \frac{\sum_{i=1}^{n} \mathbf{e}_i}{\Delta} + \sum_{i=1}^{n} x_{i,\ell}
\end{aligned}$$

If $\sum_{i=1}^{n} \mathbf{e}_i < \frac{\Delta}{2}$, then $\lceil \frac{X_\ell}{\Delta} \rfloor = \sum_{i=1}^{n} x_{i,\ell} + 1$. This shows the correctness of our algorithm.

Theorem 3. *Let κ_s and κ_c be the statistical and computational security parameters. Let L be the input dimension and n be the number of clients, that are* $\mathrm{poly}(\kappa_c)$*. Let δ be the dropout threshold and η be the corruption threshold such that $\delta + \eta < 1$. Then, there exists an efficient simulator* Sim *such that for all $\mathsf{K} \subset [n]$ such that $|\mathsf{K}| \leqslant \eta n$, inputs $X = \{\mathbf{x}_{i,\ell}\}_{i \in n \setminus \mathsf{K}}$, and for all adversaries $\mathcal{A}$ against Construction 4, that controls the server and the set of corrupted clients* K*, which behave semi-honestly (resp. maliciously), the output of* Sim *is computationally indistinguishable from the joint view of the server and the corrupted clients.* Sim *is allowed to query $F_{D,\delta}(X)^\ell$ (defined in Equation 1) once, per iteration.*

Proof. The proof proceeds similar to that of Theorem 2, through a sequence of hybrids. However, there are a few differences. Construction 2 has the error vector $\mathbf{e} \leftarrow\!\!\$\; \chi$. However, we will replace $\mathbf{e} = \mathbf{e}' + \mathbf{f}'$ where $\mathbf{e}', \mathbf{f}' \leftarrow\!\!\$\; \chi'$, the distribution present in Hint-LWE Assumption (see Definition 6). The hybrid descriptions are similar, so we only specify the differences:

- In Hybrid_2 we will set:

$$\mathbf{ct}_{n,\ell} = \mathbf{A} \cdot \mathsf{sd}_\ell - \sum_{i \in (\mathcal{C} \cap H) \setminus \{n\}} \mathbf{ct}_i + \mathbf{e}_\ell + \Delta \mathbf{x}_\ell$$

- We will argue that $\mathsf{Hybrid}_2, \mathsf{Hybrid}_3$ are indistinguishable under Hint-LWE Assumption. We will sketch the reduction now.
  - Recall that, from the Hint-LWE Challenge, we get $(\mathbf{A}, \mathbf{u}^*, \mathbf{s}^* := \mathbf{s} + \mathbf{r}, \mathbf{e}^* := \mathbf{e}' + \mathbf{f}')$.
  - As done for the `LWR` construction, we will set the $\mathbf{s}_1 + \mathbf{s}_n = \mathbf{s}^*$, the leakage on key.
  - For generating $\mathbf{ct}_{1,\ell}$ we will use $\mathbf{u}^*$, while also sampling a separate $\mathbf{f}_1 \leftarrow\!\!\$\; \chi'$. This gives us: $\mathbf{ct}_{1,\ell} = \mathbf{u}^* + \mathbf{f}'_{n,\ell} + \Delta \cdot \mathbf{x}_{1,\ell}$
  - We will set $\mathbf{ct}_{n,\ell} := \mathbf{A} \cdot \mathsf{sd}_\ell - \sum_{i \in (\mathcal{C} \cap H) \setminus \{n\}} \mathbf{ct}_i + \mathbf{e}^* + \sum_{i \in (\mathcal{C} \cap H) \setminus \{1\}} (\mathbf{e}'_{i,\ell} + \mathbf{f}'_{i,\ell}) + \Delta \cdot \mathbf{x}_\ell$
  - When $\mathbf{u}^*$ is the real sample, then $\mathbf{ct}_{1,\ell}$ satisfies Hybrid_2's definition. Meanwhile, the $\mathbf{ct}_{n,\ell}$ is also correctly simulated. Similarly, the case when it a random sample.

The proof of security against malicious server also follows that of the previous theorem. □

6.3 Security Against Malicious Clients

In this section, we will focus on a misbehaving client. Observe that the client can send inconsistent shares to the committee. A standard approach would be for the client to rely on verifiable secret sharing which would empower each committee member to verify if the share received by the committee member is consistent with the commitments to the polynomial that was sent by the client. However, this requires each client to perform $n \cdot \mathsf{r}$ exponentiations which can be expensive. Instead, we take the approach of a modified publicly verifiable secret sharing scheme. This approach empowers the server, which has more computation power, to verify if the sharing is consistent.

Our public verifiability will rely on a modification of SCRAPE [27]. SCRAPE test is done to check if $(\mathsf{sd}_{i,\ell}^{(1)}, \ldots, \mathsf{sd}_{i,\ell}^{(\mathsf{m})})$ is a Shamir sharing over $\mathbb{F}$ of degree $d = \mathsf{r} - 1$ (namely there exists a polynomial p of degree $\leqslant d$ such that $p(i) = s_i$ for $i = 1, \ldots, n$), one can sample $w^{(1)}, \ldots, w^{(\mathsf{m})}$ uniformly from the dual code to the Reed-Solomon code formed by the evaluations of polynomials of degree $\leqslant d$, and check if $\sum_{i=1}^{\mathsf{m}} w^{(i)} \cdot \mathsf{sd}_{i,\ell}^{(i)} = 0$ in $\mathbb{F}$. If the test passes, then $\mathsf{sd}_{i,\ell}^{(1)}, \ldots, \mathsf{sd}_{i,\ell}^{(\mathsf{m})}$ are Shamir Shares, except with probability $1/|\mathbb{F}|$.

Lemma 4 (SCRAPE Test [28])**.** *Let $\mathbb{F}$ be a finite field and let $d = \mathsf{r}-1, \mathsf{m}$ be parameters of the Shamir's Secret Sharing scheme such that $0 \leqslant d \leqslant \mathsf{m}-2$, and inputs $\mathsf{sd}_{i,\ell}^{(1)}, \ldots, \mathsf{sd}_{i,\ell}^{(\mathsf{m})} \in \mathbb{F}$. Define $v_i := \prod_{j \in [\mathsf{m}] \setminus \{i\}} (i-j)^{-1}$ and let $m^*(X) := \sum_{i=0}^{\mathsf{m}-d-2} m_i \cdot X^i \leftarrow_\$ \mathbb{F}[X]_{\leqslant \mathsf{m}-d-2}$ (i.e., a random polynomial over the field of degree at most $\mathsf{m}-d-2$). Now, let $\mathbf{w} := (v_1 \cdot m^*(1), \ldots, v_n \cdot m^*(n))$ and $\mathbf{s} := (\mathsf{sd}_{i,\ell}^{(1)}, \ldots, \mathsf{sd}_{i,\ell}^{(\mathsf{m})})$. Then,*

- *If there exists $p \in \mathbb{F}[X]_{\leqslant d}$ such that $\mathsf{sd}_{i,\ell}^{(i)}, = p(i)$ for all $i \in [n]$, then $\langle \mathbf{w}, \mathbf{s} \rangle = 0$.*
- *Otherwise,* $\Pr[\langle \mathbf{w}, \mathbf{s} \rangle = 0] = 1/|\mathbb{F}|$.

Typically, we compute the polynomial $m^*(X)$ by using the Fiat-Shamir transform over public values. Then, the vector $\mathbf{w}$ is a public vector. One simply has to hide the vector $\mathbf{s}$. As a result, we have proved that the shares do lie on the same polynomial. Note that in standard Shamir's Secret Sharing, we set $p(0) = \mathsf{sd}_{i,\ell}$, i.e., the secret. Therefore, we will have to perform inner product over a vector of length $\mathsf{m}+1$.

This results in the following additional steps on the part of the client, the committee member, and the server. We detail only these additional steps.

Construction 5 (Detecting Malicious Client Behavior)**.** Let $\mathcal{H} : \{0,1\}^* \to \mathbb{F}^{\mathsf{m}-d-2}$ where $d = \mathsf{r}-1$ be a hash function which is modeled as a random oracle. Let $\mathcal{H}' : \{0,1\}^* \to \mathbb{F}$ be the hash function used to generate the challenge. Let G be a group generated by g where Discrete Logarithm and DDH is hard, and is of prime order q, the same as the order of the field for Shamir Secret Sharing.

- Client i does the following:
  - Commit to $\mathsf{sd}_{i,\ell}^{(0)} = \mathsf{sd}_{i,\ell}, \mathsf{sd}_{i,\ell}^{(1)}, \ldots, \mathsf{sd}_{i,\ell}^{(\mathsf{m})}$ as $C_s^{(j)} := g^{\mathsf{sd}_{i,\ell}^{(j)}}$.
  - Generate the coefficients to polynomial $\mathsf{m}-d-2$ by using Fiat-Shamir transform. In other words, get $m_0, \ldots, m_{\mathsf{m}-d-2} \leftarrow \mathcal{H}(C_s^{(0)}, \ldots, C_s^{(\mathsf{m})})$
  - Compute $v_0, \ldots, v_{\mathsf{m}}$ as $v_i := \prod_{j \in \{0,\ldots,\mathsf{m}\} \setminus i} (i-j)^{-1}$.
  - Compute $\mathbf{w} := (v_0 \cdot m^*(0), \ldots, v_{\mathsf{m}} \cdot m^*(\mathsf{m}))$
  - Compute $\mathbf{t} := (t_0, \ldots, t_{\mathsf{m}}) \leftarrow_\$ \mathbb{F}$
  - Compute $C_t^{(0)}, \ldots, C_t^{(\mathsf{m})}$ as commitments to $t_0, \ldots, t_{\mathsf{m}}$ where $C_t^{(j)} := g^{t_j}$
  - Set $r := \langle \mathbf{t}, \mathbf{w} \rangle$
  - Compute $c := \mathcal{H}'(C_s^{(0)}, \ldots, C_s^{(\mathsf{m})}, C_t^{(0)}, \ldots, C_t^{(\mathsf{m})}, \mathbf{w}, r)$
  - Compute $z_0, \ldots, z_{\mathsf{m}}$ where $z_i := t_i + c \cdot \mathsf{sd}_{i,\ell}^{(i)}$
  - Set $\pi_i := \left(\left\{C_s^{(j)}\right\}, r, \mathbf{z} = (z_0, \ldots, z_{\mathsf{m}}), c\right)$
- Server does the following:
  - Upon receiving π_i from client i, the server parses $\pi_i := (\left\{C_s^{(j)}\right\}, r, \mathbf{z} = (z_0, \ldots, z_{\mathsf{m}}), c$

– It computes $\mathbf{w}$ (similar to how the client does it). It then computes $\langle \mathbf{w}, \mathbf{z} \rangle$. It checks to see if this is equal to the value r sent by the client i.
– For each $j = 0, \ldots, \mathsf{m}$, compute $C_t^{(j)} = g^{z_j} \cdot \left(C_s^{(j)}\right)^{-c}$
– Compute $c' = \mathcal{H}'(C_s^{(0)}, \ldots, C_s^{(\mathsf{m})}, C_t^{(0)}, \ldots, C_t^{(\mathsf{m})}, \mathbf{w}, r)$
– Accept input from client i if $c == c'$.
– The server sends $C_s^{(j)}$ to committee member j, along with the encrypted shares for committee member j.

- Committee member j does the following:

  – Decrypt and recover the share $\mathsf{sd}_{i,\ell}^{(j)}$. Verify that this matches the commitment forwarded by the server.

It is easy to verify that an honest prover will satisfy the proof. This follows from the SCRAPE test (Lemma 4) and also the completeness of the generalized Σ-protocol. Meanwhile, if there exists two accepting transcripts $\mathbf{z}_1, \mathbf{z}_2$ corresponding to c_1, c_2, then one can extract a witness for $\mathsf{sd}_{i,\ell}^{(0)}, \ldots, \mathsf{sd}_{i,\ell}^{(\mathsf{m})}$. This guarantees soundness. The zero-knowledge property follows by simply sampling a random $\mathbf{z}$ and then setting the choice of the $C_t^{(j)}$ by following the verification steps. Further, it can set $r = \langle \mathbf{z}, \mathbf{w} \rangle$.

6.4 One-shot Private Aggregation without Leakage Simulation

It is easy to observe that the server, in the previous constructions of OPA, recovers the sum of the keys. This constitutes leakage of the honest clients' keys. Consequently, our proofs relied on the pseudorandomness, even contingent on this leakage. Furthermore, these constructions allowed the generation of a new key, in every iteration.

We will now present a construction OPA' such that: (a) a client's keys can be reused across multiple iterations, and, (b) the server does not get the sum of the keys but rather a function of pseudorandom values, which can argued as itself being pseudorandom. The core technique we employ in this work is a distributed, key-homomorphic PRF. We formally present constructions from the CL framework in Section C. Specifically, we defer $\mathsf{OPA}_{\mathsf{CL}}$ to the appendix in Section C.2. Similarly, we present $\mathtt{LWR}$ based construction in Section D. Meanwhile, we broadly describe the intuition behind our construction.

A distributed-key-homomorphic PRF has two specific algorithms: Eval which allows to evaluation of the PRF with key k_i at a point. While, $\mathsf{P\text{-}Eval}$ allows the PRF to be evaluated at a share of the key $\mathsf{k}_i^{(j)}$ to get a partial evaluation. Therefore, in our construction, the clients mask a vector of inputs $\mathbf{x}_{i,\ell}$ by computing a pseudorandom evaluation of $\mathsf{DPRF.Eval}(\mathsf{k}_{i,1}, \ell, \ldots, \mathsf{DPRF.Eval}(\mathsf{k}_{i,L}, \ell)$. Meanwhile, the auxiliary information send to the committee member will be $\mathsf{P\text{-}Eval}(\mathsf{k}_{i,k}^{(j)}, \ell)$ for $k = 1, \ldots, L$ and $j \in [\mathsf{m}]$. Then, the committee members combine by multiplying the auxiliary information. The server then reconstructs on its end. Note that the server only computes $\mathsf{DPRF.Eval}(\sum_{i=1}^{n} \mathsf{k}_{i,k}, \ell)$ for $k = 1, \ldots, L$. This is a PRF evaluation and therefore the leakage is pseudorandom and can be easily simulated, replacing it with random.

7 Experiments

We benchmark $\mathsf{OPA}_{\mathsf{CL}}$ (Construction 12) and $\mathsf{OPA}_{\mathtt{LWR}}$ (Construction 3). Recall that the former is based on threshold key-homomorphic PRF and where there is no leakage simulation needed, while the latter is based on seed-homomorphic PRG. We run our experiments on an Apple M1 Pro CPU with 16 GB of unified memory, without any multi-threading or related parallelization. We use the ABIDES simulation [25] to simulate real-world network connections. ABIDES supports a latency model which is represented as a base delay and a jitter which controls the number of messages arriving within a specified time. Our base delay is set with the range from 21 microseconds to 100 microseconds), and use the default parameters for the jitter. This framework was used to measure the performance of other prior work including [68, 52]. More details on the framework can be found in [52, §G].

- Parameter Choices for $\mathsf{OPA}_{\mathtt{LWR}}$: $\mathsf{OPA}_{\mathtt{LWR}}$ is parametrized by ρ, q, p. We use the LWE-estimator [5] to estimate the security level. We follow the parameters similar to [47]. The value of $1/p$ is the error rate α in an LWE instance. Using the LWE estimator, we set $\rho := 1024$. We set q to match the field used for

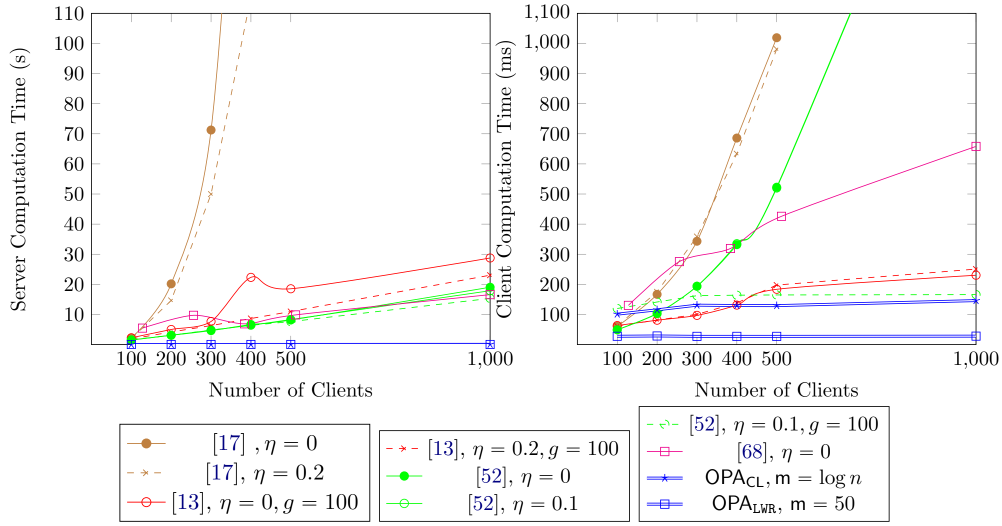


Figure 3: Client and Server Computation Time as a function of client count across different algorithms.

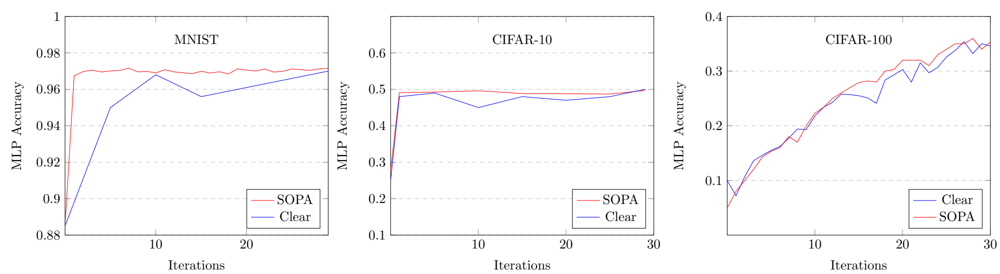


Figure 4: MLP Accuracy for different datasets: MNIST, CIFAR-10, and CIFAR-100

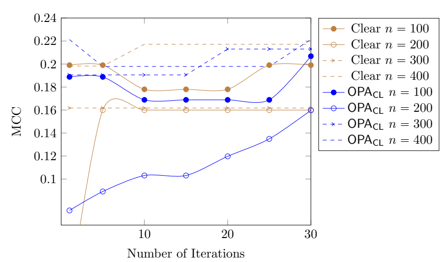


Figure 5: Measure of MCC as a function of number of iterations and number of clients, comparing $\mathsf{OPA}_{\mathsf{CL}}$ and plaintext learning.

Shamir's Secret Sharing, which is a 128-bit prime and set $p = 2^{85}$. The hardness estimated is 2^{129}. We do use Packed Secret Sharing, for benchmarking the server and client computation cost. We pack it to 64, i.e., each committee member receives 64 shares. For reconstruction to hold we set $\mathsf{m} = 50, \mathsf{r} = 34$. Assuming $\eta = \delta = 0.01$, we can estimate that setting $\mathsf{m} = 50$, then the $\Pr[\eta_C + \delta_C > 1/3] \leqslant 5 \cdot 10^{-5}$. This also satisfies the requirements of packed secret sharing which will require that $\mathsf{m} \geqslant 3/2 \cdot 1024/64$. Recall that, for packed secret sharing, we can set the corruption threshold to be $\mathsf{r} - \rho$ where ρ is the number of secrets being packed. Here ρ is 16. Note that this construction achieves committee performance independent of the vector length and is the preferred one.

- Parameter Choices for $\mathsf{OPA}_{\mathsf{CL}}$: We rely on the BICYCL library [20] and use pybind to convert the C++ code to Python. Our implementation will assume that the plaintext space is $\mathbb{Z}_p$ for a prime p. Our experiments will assume that $\mathsf{m} = \log n$.

Microbenchmarking Secure Aggregation. Our first series of experiments is to run $\mathsf{OPA}_{\mathsf{CL}}, \mathsf{OPA}_{\mathsf{LWR}}$ to build a secure aggregation protocol for $L = 1$. We also compare with existing work including [17, 13, 52, 68]. We vary the offline rates, and the ability to group clients, along with increasing the number of clients to study the performance of related work. Recall that the offline rate (denoted by η) controls the number of clients who do not participate, despite being selected. Meanwhile, we denote by g the size of the neighborhood or group. For [52], we set the input size to be bounded by 10^4. Recall that [52] does not have efficient aggregate recovery and requires input bounding. Also, [52] incurs a setup/offline client computation time of nearly 30ms, even for 100 clients. Our implementation sets $\mathsf{m} = \lfloor \log(n) \rfloor$. As can be seen from Figure 3. The key takeaway from our experiments is that our client and server performance outperforms all prior works.

Performance of $\mathsf{OPA}_{\mathsf{CL}}$. $\mathsf{OPA}_{\mathsf{CL}}$'s server running time is less than 1 second - owing to a single round protocol with support of efficient recovery of the aggregate. This outperforms all existing protocols. Each committee member computes $< 1ms$. $\mathsf{OPA}_{\mathsf{CL}}$ assumes concurrent client communication to committee and server. Note that $\mathsf{OPA}_{\mathsf{CL}}$ scales linearly with the size of the input.

Performance of $\mathsf{OPA}_{\mathsf{LWR}}$. It is to be noted that the server performance matches that of $\mathsf{OPA}_{\mathsf{CL}}$'s server performance. However, when aggregating longer vectors, which is the use-case for OPA, $\mathsf{OPA}_{\mathsf{LWR}}$ significantly outperforms $\mathsf{OPA}_{\mathsf{CL}}$, with the key savings coming from having to perform no exponentiations. Meanwhile, the client's performance in $\mathsf{OPA}_{\mathsf{LWR}}$ is much better than $\mathsf{OPA}_{\mathsf{CL}}$, again due to the lack of exponentiations. The communication cost for $\mathsf{OPA}_{\mathsf{LWR}}$ are as follows:

- Client i to Server: L field elements
- Client i to Committee: $k \cdot \mathsf{m}$ field elements where k is the number of elements being shared. With naive secret sharing, this would be 1024. Instead, when we pack into 64 different polynomials, $k = 64$.
- Committee j to Server: k field elements
- Total Sent/Received per Client: $L + k \cdot \mathsf{m}$ field elements
- Total Sent/Received by Server: $nL + \mathsf{m}k$ field elements
- Total Sent/Received per Committee Member: $n \cdot k + k$ field elements

In Section B, we describe the naive secret-sharing-based construction and also study the computation cost of $\mathsf{OPA}_{\mathsf{LWR}}$ with respect to this naive construction by micro-benchmarking the running times. Our experiments indicate that the running time of the committee member is the best with $\mathsf{OPA}_{\mathsf{LWR}}$, when compared to the naive solution. This is in addition to the improvement in the communication cost. Recall that the motivating use case for avoiding the naive solution is improved performance of the committee members, who are often other clients.

Benchmarking FL Models. To demonstrate $\mathsf{OPA}_{\mathsf{CL}}$'s viability for federated learning, we train a logistic regression model on two skewed datasets. We show that $\mathsf{OPA}_{\mathsf{CL}}$ performs very close to learning in the clear, indicating feasibility for machine learning. As our goal was to show feasibility, our experiments use one committee member for all n clients. We vary n and the number of iterations for model convergence, measuring accuracy and Matthew's Correlation Coefficient (MCC) [69] which better evaluates binary classification with unbalanced classes.

- Adult Census Dataset: We first run experiments on the adult census income dataset from [26, 55] to predict if an individual earns over $50,000 per year. The preprocessed dataset has 105 features and 45,222 records with a 25% positive class. We randomly split into training and testing, with further splitting by the clients. First, we train in the clear with weights sent to the server to aggregate. With 100 clients and 50 iterations, we achieve 82.85% accuracy and 0.51 MCC. We repeat with $\mathsf{OPA}_{\mathsf{CL}}$, one committee member, and 100 clients. With 10 iterations, we achieve 82.38% accuracy and 0.48 MCC. With 20 iterations, we achieve 82% accuracy and 0.51 MCC. Our quantization technique divides weights into integer and decimal parts (2 integer and 8 decimal values per weight). Training with 50 clients takes under 1 minute per client per iteration with no accuracy loss. This quantization yields a vector size of 1050 (10 per feature).

- We use the Kaggle Credit Card Fraud dataset [76], comprising 26 transformed principal components and amount and time features. We omit time and use the raw amount, adding an intercept. The goal is to predict if a transaction was indeed fraudulent or not. There are 30 features and 284,807 rows, with <0.2% fraudulent. Weights are multiplied by 10,000 and rounded to an integer, accounted for in aggregation. Figure 5 shows $\mathsf{OPA}_{\mathsf{CL}}$'s MCC versus clear learning for varying clients and iterations. With the accuracy multiplier, $\mathsf{OPA}_{\mathsf{CL}}$'s MCC is very close to clear learning and even outperforms sometimes. The highly unbalanced dataset demonstrates $\mathsf{OPA}_{\mathsf{CL}}$ can achieve strong performance even in challenging real-world scenarios.
- We then train a vanilla multi-layer perceptron (MLP) classifier on three datasets: MNIST, CIFAR-10, CIFAR-100. We quantize the weights by multiplying with 2^{16}. The MLP accuracy, as a function of the iteration count, is plotted in Figure 4. Our experiments demonstrate that $\mathsf{OPA}_{\mathsf{CL}}$ preserves accuracy while ensuring the privacy of client data. Note that vanilla MLP classifiers do not typically offer good performance for CIFAR datasets, but note that the goal of our experiments was to show that $\mathsf{OPA}_{\mathsf{CL}}$ does not impact accuracy.

References

[1] Damiano Abram, Ivan Damgård, Claudio Orlandi, and Peter Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022, Part IV*, volume 13510 of *Lecture Notes in Computer Science*, pages 421–452. Springer, Cham, August 2022.

[2] Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In Eike Kiltz and Vinod Vaikuntanathan, editors, *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part II*, volume 13748 of *Lecture Notes in Computer Science*, pages 502–531. Springer, 2022.

[3] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In Clemente Galdi and Stanislaw Jarecki, editors, *Security and Cryptography for Networks - 13th International Conference, SCN 2022, Amalfi, Italy, September 12-14, 2022, Proceedings*, volume 13409 of *Lecture Notes in Computer Science*, pages 516–539. Springer, 2022.

[4] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or fuzzy ibe) from lattices. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012: 15th International Conference on Theory and Practice of Public Key Cryptography*, volume 7293 of *Lecture Notes in Computer Science*, pages 280–297. Springer, Berlin, Heidelberg, May 2012.

[5] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[6] Apple and Google. Exposure notification privacy-preserving analytics (ENPA), 2021.

[7] Arasu Arun, Chaya Ganesh, Satya V. Lokam, Tushar Mopuri, and Sriram Sridhar. Dew: A transparent constant-sized polynomial commitment scheme. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 13941 of *Lecture Notes in Computer Science*, pages 542–571. Springer, Cham, May 2023.

[8] Thomas Attema, Ignacio Cascudo, Ronald Cramer, Ivan Bjerre Damgård, and Daniel Escudero. Vector commitments over rings and compressed σ-protocols. Cryptology ePrint Archive, Report 2022/181, 2022.

[9] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 719–737. Springer, Berlin, Heidelberg, April 2012.

[10] Laasya Bangalore, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, and Muthuramakrishnan Venkitasubramaniam. Flag: A framework for lightweight robust secure aggregation. In Joseph K. Liu, Yang Xiang, Surya Nepal, and Gene Tsudik, editors, *ASIACCS 23: 18th ACM Symposium on Information, Computer and Communications Security*, pages 14–28. ACM Press, July 2023.

[11] Daniela Becker, Jorge Guajardo, and Karl-Heinz Zimmermann. Revisiting private stream aggregation: Lattice-based PSA. In *ISOC Network and Distributed System Security Symposium – NDSS 2018*. The Internet Society, February 2018.

[12] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. ACORN: Input validation for secure aggregation. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 4805–4822, Anaheim, CA, August 2023. USENIX Association.

[13] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1253–1269, 2020.

[14] James Bell-Clark, Adrià Gascón, Baiyu Li, Mariyana Raykova, Phillipp Schoppmann, and Pierre Tholoniat. Willow: Secure aggregation with one-shot clients. Theory and Practice of Multi-Party Computation Workshop, 2024.

[15] Fabrice Benhamouda, Marc Joye, and Beno$\hat{3}$T Libert. A new framework for privacy-preserving aggregation of time-series data. *ACM Trans. Inf. Syst. Secur.*, 18(3), mar 2016.

[16] Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. On linear communication complexity for (maximally) fluid MPC. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 263–294. Springer, Cham, August 2023.

[17] 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 Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191. ACM, 2017.

[18] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to IOPs and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 561–586. Springer, Cham, August 2019.

[19] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 410–428. Springer, Berlin, Heidelberg, August 2013.

[20] Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. I want to ride my BICYCL : BICYCL implements cryptography in class groups. *J. Cryptol.*, 36(3):17, 2023.

[21] Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 613–645. Springer, Cham, August 2023.

[22] Joakim Brorsson and Martin Gunnarsson. Dipsauce: Efficient private stream aggregation without trusted parties. To Appear in NordSec23, 2023.

[23] Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, 1(2):107–118, June 1988.

[24] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent SNARKs from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, Cham, May 2020.

[25] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. Abides: Towards high-fidelity multi-agent market simulation. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, SIGSIM-PADS '20, page 11–22, New York, NY, USA, 2020. Association for Computing Machinery.

[26] David Byrd, Vaikkunth Mugunthan, Antigoni Polychroniadou, and Tucker Balch. Collusion resistant federated learning with oblivious distributed differential privacy. In *Proceedings of the Third ACM International Conference on AI in Finance*, ICAIF '22, page 114–122, New York, NY, USA, 2022. Association for Computing Machinery.

[27] Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17: 15th International Conference on Applied Cryptography and Network Security*, volume 10355 of *Lecture Notes in Computer Science*, pages 537–556. Springer, Cham, July 2017.

[28] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 216–248. Springer, Cham, May 2024.

[29] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ECDSA from hash proof systems and efficient instantiations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part III*, volume 11694 of *Lecture Notes in Computer Science*, pages 191–221. Springer, Cham, August 2019.

[30] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Bandwidth-efficient threshold EC-DSA. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020: 23rd International Conference on Theory and Practice of Public Key Cryptography, Part II*, volume 12111 of *Lecture Notes in Computer Science*, pages 266–296. Springer, Cham, May 2020.

[31] Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. Encryption switching protocols revisited: Switching modulo p. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part I*, volume 10401 of *Lecture Notes in Computer Science*, pages 255–287. Springer, Cham, August 2017.

[32] Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from DDH. In Kaisa Nyberg, editor, *Topics in Cryptology – CT-RSA 2015*, volume 9048 of *Lecture Notes in Computer Science*, pages 487–505. Springer, Cham, April 2015.

[33] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Practical fully secure unrestricted inner product functional encryption modulo p. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 733–764. Springer, Cham, December 2018.

[34] Pyrros Chaidos and Geoffroy Couteau. Efficient designated-verifier non-interactive zero-knowledge proofs of knowledge. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part III*, volume 10822 of *Lecture Notes in Computer Science*, pages 193–221. Springer, Cham, April / May 2018.

[35] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Joohee Lee, Junbum Shin, and Yongsoo Song. Lattice-based secure biometric authentication for hamming distance. In Joonsang Baek and Sushmita Ruj, editors, *ACISP 21: 26th Australasian Conference on Information Security and Privacy*, volume 13083 of *Lecture Notes in Computer Science*, pages 653–672. Springer, Cham, December 2021.

[36] Kevin Choi, Aathira Manoj, and Joseph Bonneau. SoK: Distributed randomness beacons. In *2023 IEEE Symposium on Security and Privacy*, pages 75–92. IEEE Computer Society Press, May 2023.

[37] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 94–123, Virtual Event, August 2021. Springer, Cham.

[38] Jean-Sébastien Coron. On the exact security of full domain hash. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 229–235. Springer, Berlin, Heidelberg, August 2000.

[39] Henry Corrigan-Gibbs. Privacy-preserving firefox telemetry with prio, 2020.

[40] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In Aditya Akella and Jon Howell, editors, *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 259–282. USENIX Association, 2017.

[41] Geoffroy Couteau, Dahmun Goudarzi, Michael Klooß, and Michael Reichle. Sharp: Short relaxed range proofs. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 609–622. ACM Press, November 2022.

[42] Geoffroy Couteau, Michael Klooß, Huang Lin, and Michael Reichle. Efficient range proofs with transparent setup from bounded integer commitments. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part III*, volume 12698 of *Lecture Notes in Computer Science*, pages 247–277. Springer, Cham, October 2021.

[43] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006: 9th International Conference on Theory and Practice of Public Key Cryptography*, volume 3958 of *Lecture Notes in Computer Science*, pages 75–90. Springer, Berlin, Heidelberg, April 2006.

[44] Yi Deng, Shunli Ma, Xinxuan Zhang, Hailong Wang, Xuyang Song, and Xiang Xie. Promise Σ-protocol: How to construct efficient threshold ECDSA from encryptions based on class groups. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 557–586. Springer, Cham, December 2021.

[45] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pages 417–446. Springer, Cham, April 2023.

[46] Drand. Drand/drand: a distributed randomness beacon daemon - go implementation.

[47] Johannes Ernst and Alexander Koch. Private stream aggregation with labels in the standard model. *Proc. Priv. Enhancing Technol.*, 2021(4):117–138, 2021.

[48] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *24th Annual ACM Symposium on Theory of Computing*, pages 699–710. ACM Press, May 1992.

[49] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658. Springer, 2014.

[50] Noemi Glaeser, Matteo Maffei, Giulio Malavolta, Pedro Moreno-Sanchez, Erkan Tairi, and Sri AravindaKrishnan Thyagarajan. Foundations of coin mixing services. Cryptology ePrint Archive, Report 2022/942, 2022.

[51] Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, USA, 2004.

[52] Yue Guo, Antigoni Polychroniadou, Elaine Shi, David Byrd, and Tucker Balch. MicroFedML: Privacy preserving federated learning for small weights. Cryptology ePrint Archive, Report 2022/714, 2022.

[53] Shai Halevi, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. Additive randomized encodings and their applications. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 203–235. Springer, 2023.

[54] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2011.

[55] Bargav Jayaraman, Lingxiao Wang, David Evans, and Quanquan Gu. Distributed learning without distress: privacy-preserving empirical risk minimization. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6346–6357, Red Hook, NY, USA, 2018. Curran Associates Inc.

[56] Marc Joye and Beno3̂t Libert. A scalable scheme for privacy-preserving aggregation of time-series data. In Ahmad-Reza Sadeghi, editor, *FC 2013: 17th International Conference on Financial Cryptography and Data Security*, volume 7859 of *Lecture Notes in Computer Science*, pages 111–125. Springer, Berlin, Heidelberg, April 2013.

[57] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Za3̈d Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Found. Trends Mach. Learn.*, 14(1-2):1–210, 2021.

[58] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive VSS using class groups and application to DKG. Cryptology ePrint Archive, Report 2023/451, 2023.

[59] Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 549–580. Springer, Cham, August 2023.

[60] Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 530–560. Springer, Cham, August 2019.

[61] Iraklis Leontiadis, Kaoutar Elkhiyaoui, and Refik Molva. Private and dynamic time-series data aggregation with trust relaxation. In Dimitris Gritzalis, Aggelos Kiayias, and Ioannis G. Askoxylakis, editors, *CANS 14: 13th International Conference on Cryptology and Network Security*, volume 8813 of *Lecture Notes in Computer Science*, pages 305–320. Springer, Cham, October 2014.

[62] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. LERNA: Secure single-server aggregation via key-homomorphic masking. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part I*, volume 14438 of *Lecture Notes in Computer Science*, pages 302–334. Springer, Singapore, December 2023.

[63] Yehuda Lindell. *How to Simulate It – A Tutorial on the Simulation Proof Technique*, pages 277–346. Springer International Publishing, Cham, 2017.

[64] Helger Lipmaa. Secure accumulators from euclidean rings without trusted setup. In Feng Bao, Pierangela Samarati, and Jianying Zhou, editors, *ACNS 12: 10th International Conference on Applied Cryptography and Network Security*, volume 7341 of *Lecture Notes in Computer Science*, pages 224–240. Springer, Berlin, Heidelberg, June 2012.

[65] Zizhen Liu, Si Chen, Jing Ye, Junfeng Fan, Huawei Li, and Xiaowei Li. SASH: efficient secure aggregation based on SHPRG for federated learning. In James Cussens and Kun Zhang, editors, *Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands*, volume 180 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR, 2022.

[66] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. RoFL: Robustness of secure federated learning. In *2023 IEEE Symposium on Security and Privacy*, pages 453–476. IEEE Computer Society Press, May 2023.

[67] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010.

[68] 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 *2023 IEEE Symposium on Security and Privacy*, pages 477–496. IEEE Computer Society Press, May 2023.

[69] B.W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) - Protein Structure*, 405(2):442–451, 1975.

[70] K. MCCURLEY. Cryptographic key distribution and computation in class groups. *Proceedings of NATO ASI Number Theory and applications*, pages 459–479, 1989.

[71] Moni Naor, Benny Pinkas, and Omer Reingold. Distributed pseudo-random functions and KDCs. In Jacques Stern, editor, *Advances in Cryptology – EUROCRYPT'99*, volume 1592 of *Lecture Notes in Computer Science*, pages 327–346. Springer, Berlin, Heidelberg, May 1999.

[72] Dinh Duy Nguyen, Duong Hieu Phan, and David Pointcheval. Verifiable decentralized multi-client functional encryption for inner product. In Jian Guo and Ron Steinfeld, editors, *Advances in Cryptology – ASIACRYPT 2023, Part V*, volume 14442 of *Lecture Notes in Computer Science*, pages 33–65. Springer, Singapore, December 2023.

[73] Johannes Ottenhues and Alexander Koch. LaPSuS - A lattice-based private stream aggregation scheme under scrutiny. In Clemente Galdi and Duong Hieu Phan, editors, *SCN 24: 14th International Conference on Security in Communication Networks, Part II*, volume 14974 of *Lecture Notes in Computer Science*, pages 228–248. Springer, Cham, September 2024.

[74] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 2429–2443. ACM Press, November 2022.

[75] Christopher Patton, Richard Barnes, and Phillipp Schoppmann. Verifiable Distributed Aggregation Functions. Internet-Draft draft-patton-cfrg-vdaf-01, Internet Engineering Task Force, March 2022. Work in Progress.

[76] Andrea Dal Pozzolo, Olivier Caelen, Reid A. Johnson, and Gianluca Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166, 2015.

[77] Mayank Raikwar and Danilo Gligoroski. SoK: Decentralized randomness beacon protocols. In Khoa Nguyen, Guomin Yang, Fuchun Guo, and Willy Susilo, editors, *ACISP 22: 27th Australasian Conference on Information Security and Privacy*, volume 13494 of *Lecture Notes in Computer Science*, pages 420–446. Springer, Cham, November 2022.

[78] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1961–1979, 2023.

[79] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. ELSA: Secure aggregation for federated learning with malicious actors. In *2023 IEEE Symposium on Security and Privacy*, pages 1961–1979. IEEE Computer Society Press, May 2023.

[80] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), sep 2009.

[81] István András Seres, Péter Burcsi, and Péter Kutas. How (not) to hash into class groups of imaginary quadratic fields? Cryptology ePrint Archive, Report 2024/034, 2024.

[82] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[83] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *ISOC Network and Distributed System Security Symposium – NDSS 2011*. The Internet Society, February 2011.

[84] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, Berlin, Heidelberg, May 2000.

[85] Jonathan Takeshita, Zachariah Carmichael, Ryan Karl, and Taeho Jung. TERSE: tiny encryptions and really speedy execution for post-quantum private stream aggregation. In Fengjun Li, Kaitai Liang, Zhiqiang Lin, and Sokratis K. Katsikas, editors, *Security and Privacy in Communication Networks - 18th EAI International Conference, SecureComm 2022, Virtual Event, October 2022, Proceedings*, volume 462 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 331–352. Springer, 2022.

[86] Jonathan Takeshita, Ryan Karl, Ting Gong, and Taeho Jung. SLAP: Simple lattice-based private stream aggregation protocol. Cryptology ePrint Archive, Report 2020/1611, 2020.

[87] Jonathan Takeshita, Ryan Karl, Ting Gong, and Taeho Jung. SLAP: Simpler, improved private stream aggregation from ring learning with errors. *Journal of Cryptology*, 36(2):8, April 2023.

[88] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabien Laguillaumie, and Giulio Malavolta. Efficient CCA timed commitments in class groups. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2663–2684. ACM Press, November 2021.

[89] Ida Tucker. *Functional encryption and distributed signatures based on projective hash functions, the benefit of class groups*. Theses, Université de Lyon, October 2020.

[90] Hendrik Waldner, Tilen Marc, Miha Stopar, and Michel Abdalla. Private stream aggregation from labeled secret sharing schemes. Cryptology ePrint Archive, Report 2021/081, 2021.

[91] Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019, Part III*, volume 11478 of *Lecture Notes in Computer Science*, pages 379–407. Springer, Cham, May 2019.

[92] Benjamin Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, October 2020.

[93] Tsz Hon Yuen, Handong Cui, and Xiang Xie. Compact zero-knowledge proofs for threshold ECDSA with trustless setup. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 481–511. Springer, Cham, May 2021.

[94] Yulin Zhao, Hualin Zhou, and Zhiguo Wan. SuperFL: Privacy-preserving federated learning with efficiency and robustness. Cryptology ePrint Archive, Report 2024/081, 2024.

A Cryptographic Preliminaries

For completeness, we discuss secret sharing in Section A.1. We discuss pseudorandom functions in Section A.3. We then introduce lattice-based cryptographic assumptions in Section A.2.

A.1 Secret Sharing

A key component of threshold cryptography is the ability to compute distributed exponentiation by sharing a secret. More formally, the standard approach is to compute g^s for some $g \in \mathbb{G}$ where $\mathbb{G}$ is a finite group and s is a secret exponent that has been secret-shared among multiple parties. This problem is much simpler when you assume that the group order is a publicly known prime p which then requires you to share the secret over the field $\mathbb{Z}_p$. This was the observation of Shamir [82] whereby a secret s can be written as a linear combination of $\sum_{i \in \mathcal{S}} \alpha_i \mathsf{s}_i \bmod p$ where $\mathcal{S}$ is a set of servers that is sufficiently large and holds shares of the secret s_i and α_i is only a function of the indices in $\mathcal{S}$. It follows that if each server provides $g_i = g^{\mathsf{s}_i}$, then one can compute $g^s = g^{\sum_{i \in \mathcal{S}} \alpha_i \cdot \mathsf{s}_i} = \prod_{i \in \mathcal{S}} g_i^{\alpha_i}$. Formally, this is defined below.

Construction 6 (Shamir's Secret Sharing over $\mathbb{F}_q$)**.** Consider the following $(\mathsf{t}, \mathsf{r}, \mathsf{m})$ Secret Sharing Scheme where m is the total number of parties, t is the corruption threshold, r is the threshold for reconstruction. Then, we have the following scheme:

- $\mathsf{Share}(\mathsf{s}, \mathsf{t}, \mathsf{r}, \mathsf{m})$: Sample a random polynomial $f(X) \in \mathbb{F}_q[X]$ of degree $\mathsf{r} - 1$ such that $f(0) = \mathsf{s}$. Then, **return** $\{\mathsf{s}^{(j)} := f(j)\}_{j \in [\mathsf{m}]}$
- $\mathsf{Coeff}(\mathcal{S})$: On input of a set $\mathcal{S} = \{i_1, \ldots, i_\mathsf{r}, \ldots\} \subseteq [\mathsf{m}]$ of at least r indices, compute $\lambda_{i_j} = \prod_{\zeta \in [\mathsf{r}] \setminus \{j\}} \frac{i_\zeta}{i_\zeta - i_j}$. Then, **return** $\{\lambda_{i_j}\}_{i_j \in \{i_1, \ldots, i_\mathsf{r}\}}$
- $\mathsf{Reconstruct}(\{s^{(j)}\}_{j \in \mathcal{S}})$: If $|\mathcal{S}| \geqslant \mathsf{r}$, then output $\sum_{j \in \mathcal{S}} \lambda_j \cdot \mathsf{s}^{(j)}$ where $\{\lambda_j\} \leftarrow \mathsf{Coeff}(\mathcal{S})$.

The correctness of the scheme guarantees that the secret s is correctly reconstructed.

Construction 7 (Packed Secret Sharing over $\mathbb{F}_q$)**.** Consider the following $(\mathsf{t}, \mathsf{r}, \mathsf{m})$ Secret Sharing Scheme where m is the total number of parties, t is the corruption threshold, r is the threshold for reconstruction. Further, let ρ be the number of secrets being packed which are to be embedded at points $\mathsf{pos}_1, \ldots, \mathsf{pos}_\rho$ where $\mathsf{pos}_i = \mathsf{m} + i$. Here, $\mathsf{t} := \mathsf{r} - \rho$. Then, we have the following scheme:

$\mathsf{PShare}(\mathbf{s} = (\mathsf{s}_1, \ldots, \mathsf{s}_\rho), \mathsf{t}, \mathsf{r}, \mathsf{m})$

$(r_0, \ldots, r_{\mathsf{r}-\rho-1}) \leftarrow\$ \mathbb{F}_q$

$q(X) := \sum_{i=0}^{\mathsf{r}-\rho-1} X^i \cdot r_i$

$\mathsf{pos}_i = \mathsf{m} + i$ **for** $i = 1, \ldots, \rho$

for $i \in [\rho]$ **do**

$\quad L_i(X) := \prod_{j \in [\rho] \setminus i} \frac{X - \mathsf{pos}_j}{\mathsf{pos}_i - \mathsf{pos}_j} \cdot \Delta$

$f(X) := q(X) \prod_{i=1}^{\rho} (X - \mathsf{pos}_i) + \sum_{i=1}^{\rho} \mathsf{s}_i \cdot L_i(X)$

return $\{\mathsf{s}^{(i)}\}_{i \in [\mathsf{m}]}$

$\mathsf{Reconstruct}(\{\mathsf{s}^{(i)}\}_{i \in \mathcal{S}})$

if $|\mathcal{S}| < \mathsf{r}$ **return** $\perp$

Parse $\mathcal{S} := \{i_1, \ldots, i_\mathsf{t}, \ldots\}$

for $k \in [\rho]$

$\quad$ **for** $j \in [\mathsf{t}]$

$\quad\quad \Lambda_{i_j}(X) := \prod_{\zeta \in [\mathsf{t}] \setminus j} \frac{i_\zeta - X}{i_\zeta - i_j}$

$\quad\quad \mathsf{s}'_k := \sum_{j \in [t]} \Lambda_{i_j}(m + k) \cdot \mathsf{s}^{(j)}$

$\quad$ **return** $\mathbf{s}' := (\mathsf{s}'_1, \ldots, \mathsf{s}'_\rho)$

Parameters.

- For reconstruction, we require $\mathsf{r} < \mathsf{m}(1 - \delta_C)$ where δ_C is the dropout rate within the committee.
- For security, we require that $(\mathsf{r} - \rho) > \mathsf{m} \cdot \eta_C$ where η_C is the corruption rate.

Combining, we get $\mathsf{m} > \rho/(1 - \delta_C - \eta_C)$. Recall that we need $\delta_C + \eta_C < 1/3$, for byzantine fault tolerance. In other words, setting $\mathsf{m} \geqslant 3\rho/2$ is sufficient.

Remark 3 (Optimizations for Packed Secret Sharing)**.** Observe that the polynomial $L_i(X)$ is only dependent on points pos_i, which are the points where the secrets are embedded. This can be pre-processed, and indeed, can be a part of the setup algorithm which distributes it to all the clients. Furthermore, rather than naively reconstructing the Lagrange polynomial, one can also rely on FFT techniques to achieve speed up.

Unfortunately, the above protocols do not extend to settings where the order of the group is not prime, not publicly known, or even possibly unknown to everyone. In this setting, the work of Damgård and Thorbek presents a construction to build Linear Integer Secret Sharing (LISS) schemes. In this work, we rely on the simpler scheme that extends Shamir's secret sharing into the integer setting from the work of Braun *et al.* [21]. We also extend the Packed Secret Sharing scheme to this integer setting in Construction 9.

Definition 2 (Secret Sharing over $\mathbb{Z}$)**.** *A* $(\mathsf{t},\mathsf{r},\mathsf{m})$ *Linear Integer Secret Sharing Scheme* LISS *is a tuple of PPT algorithms* $\mathsf{LISS} := (\mathsf{Share}, \mathsf{GetCoeff}, \mathsf{Reconstruct})$*, with the following public parameters: the statistical security parameter* κ_s*, the number of parties* m*, the corruption threshold* t*, and reconstruction threshold* r *of secrets needed for reconstruction, the randomness bit length* ℓ_r*, the bit length of the secret* ℓ_s *and the offset by which the secret is multiplied, denoted by* $\Delta = \mathsf{m}!$*, and the following syntax:*

- $(\mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{m}) \leftarrow\$ \mathsf{Share}(\mathsf{s}, \mathsf{m}, \mathsf{r}, \mathsf{t})$*: On input of the secret* s*, the number of parties* m*, and the threshold* t*, the share algorithm outputs shares* $\mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{m}$ *such that party* i *receives* s_i*.*
- $\{\lambda_i\}_{i\in\mathcal{S}} \leftarrow \mathsf{GetCoeff}(\mathcal{S})$*: On input of a set* $\mathcal{S}$ *of at least* r *indices, the* $\mathsf{GetCoeff}$ *algorithm outputs the set of coefficients for polynomial reconstruction.*
- $\mathsf{s}' \leftarrow \mathsf{Reconstruct}(\{\mathsf{s}_i\}_{i\in\mathcal{S}})$*: On input of a set of secrets of at least* r *shares, the reconstruction algorithm outputs the secret* s'*.*

We further require the following security properties.

- *Correctness: For any* $\mathsf{m}, \kappa_s, \mathsf{t}, \mathsf{r}, \ell_s, \ell_r \in \mathbb{Z}$ *with* $\mathsf{t} < \mathsf{r} \leqslant \mathsf{m}$*, and any set* $\mathcal{S} \subseteq [m]$ *with* $|\mathcal{S}| \geqslant \mathsf{r}$*, for any* $\mathsf{s} \in \mathbb{Z}$ *such that* $\mathsf{s} \in [0, 2^{\ell_s})$ *the following holds:*
$$\Pr\left[\mathsf{s}' = f(\mathsf{s}) \;\middle|\; \begin{array}{c}(\mathsf{s}_1, \ldots, \mathsf{s}_m) \leftarrow\$ \mathsf{Share}(\mathsf{s}, \mathsf{m}, \mathsf{r}, \mathsf{t}) \\ \mathsf{s}' \leftarrow \mathsf{Reconstruct}(\{\mathsf{s}_i\}_{i\in\mathcal{S}})\end{array}\right]$$
where f *is some publicly computable function, usually* $f(\mathsf{s}) = \Delta^2 \cdot \mathsf{s}$.
- *Statistical Privacy [43]: We say that a* $(\mathsf{t},\mathsf{r},\mathsf{m})$ *linear integer secret sharing scheme* LISS *is statistically private if for any set of corrupted parties* $\mathcal{C} \subset [\mathsf{m}]$ *with* $|\mathcal{C}| \leqslant \mathsf{t}$*, and any two secrets* $\mathsf{s}, \mathsf{s}' \in [0, 2^{\ell_s})$ *and for independent random coins* ρ, ρ' *such that* $\{\mathsf{s}_i\}_{i\in[\mathsf{m}]} \leftarrow\$ \mathsf{Share}(\mathsf{s}; \rho)$*,* $\{\mathsf{s}_i\}'_{i\in[\mathsf{m}]} \leftarrow\$ \mathsf{Share}(\mathsf{s}'; \rho')$ *we have that the statistical distance between:* $\{\mathsf{s}_i | i \in \mathcal{C}\}$ *and* $\{\mathsf{s}'_i | i \in \mathcal{C}\}$ *is negligible in the statistical security parameter* κ_s*.*

Construction 8 (Shamir's Secret Sharing over $\mathbb{Z}$)**.** Consider the following $(\mathsf{t},\mathsf{r},\mathsf{m})$ Integer Secret Sharing scheme where m is the number of parties, t is the corruption threshold, and r is the threshold for reconstruction. Further, let κ_s be a statistical security parameter. Let ℓ_s be the bit length of the secret and let ℓ_r be the bit length of the randomness. Then, we have the following scheme:

$\mathsf{Share}(\mathsf{s},\mathsf{t},\mathsf{r},\mathsf{m})$	$\mathsf{GetCoeff}(\mathcal{S})$	$\mathsf{Reconstruct}(\{\mathsf{s}^{(i)}\}_{i\in\mathcal{S}})$
$\Delta := \mathsf{m}!, \tilde{\mathsf{s}} := \mathsf{s} \cdot \Delta$	**if** $\lvert\mathcal{S}\rvert \geqslant \mathsf{r}$	**if** $\lvert\mathcal{S}\rvert \geqslant \mathsf{r}$
$(r_1, \ldots, r_{\mathsf{r}-1}) \leftarrow\$ [0, 2^{\ell_r+\kappa_s})$	**for** $i \in \mathcal{S}$ **do**	$\{\Lambda_i\}_{i\in\mathcal{S}} \leftarrow \mathsf{GetCoeff}(\mathcal{S})$
$f(X) := \tilde{\mathsf{s}} + \sum_{i=1}^{\mathsf{r}-1} r_i \cdot X^i$	$\Lambda_i := \prod_{j\in\mathcal{S}\setminus\{i\}} \frac{x_j}{x_j - x_i} \cdot \Delta$	$\mathsf{s}' := \sum_{i\in\mathcal{S}} \Lambda_i \cdot \mathsf{s}^{(i)}$
return $\left\{\mathsf{s}^{(i)} = f(i)\right\}_{i\in[\mathsf{m}]}$	**return** $\{\Lambda_i\}_{i\in\mathcal{S}}$	**return** s'

We omit the proof of correctness as it is similar to the original Shamir's Secret Sharing scheme. However, we highlight the critical differences:

- Unlike Shamir's Secret Sharing over fields, the secret here is already multiplied by the offset Δ. Therefore, any attempt to reconstruct can only yield $\mathsf{s} \cdot \Delta$

- However, note that the inverse of $x_j - x_i$ which was defined over the field $\mathbb{Z}_q$ might not exist or be efficiently computable in a field of unknown order. Instead, we multiply the Lagrange coefficients by Δ. Consequently, the reconstruction yields $\Delta \cdot \tilde{\mathsf{s}}$ which equals $\mathsf{s} \cdot \Delta^2$.

Theorem 5 ([21])**.** *Construction 8 is statistically private provided* $\ell_r \geqslant \ell_s + \lceil \log_2(h_{\max} \cdot (\mathsf{t}-1)) \rceil + 1$ *where* $h_{\max}$ *is an upper bound on the coefficients of the sweeping polynomial.*

We refer the readers to the proof in [21, §B.1]. The key idea behind the proof is first to show that there exists a "sweeping polynomial" such that at each of the points that the adversary has a share of, the polynomial evaluates to 0 while at the point where the secret exists, it contains the offset Δ. Implicitly, one can add the sweeping polynomial to the original polynomial whereby the sweeping polynomial "sweeps" away the secret information that the adversary has gained knowledge of. Meanwhile, in the later section, we present the proof for the generic construction that uses Shamir's Packed Secret Sharing over the integer space. This again uses the idea of a sweeping polynomial.

Construction 9 (Shamir's Packed Secret Sharing over $\mathbb{Z}$)**.** Let m be the number of parties and ρ be the number of secrets that are packed in one sharing. Further, let t denote the threshold for reconstruction (implies that corruption threshold is $\mathsf{t} - \rho$). Then, consider the following $(\mathsf{m}, \mathsf{t}, \rho)$ Integer Secret Sharing Scheme with system parameters κ_s as the statistical security parameter, ℓ_s is the bit length of the a secret, and let ℓ_r be the bit length of the randomness. Then, we have the following scheme:

$\mathsf{PackedShare}(\mathbf{s} = (\mathsf{s}_1, \ldots, \mathsf{s}_\rho), \mathsf{t}, \mathsf{m})$

$\Delta := \mathsf{m}!, \tilde{\mathbf{s}} := \mathbf{s} \cdot \Delta$

$(r_0, \ldots, r_{\mathsf{t}-\rho-1}) \leftarrow\!\!\$\ [0, 2^{\ell_r + \kappa_s})$

$$q(X) := \sum_{i=0}^{\mathsf{t}-\rho-1} X^i \cdot r_i$$

$\mathsf{pos}_i = \mathsf{m} + i$ **for** $i = 1, \ldots, \rho$

for $i \in [\rho]$ **do**

$$L_i(X) := \prod_{j \in [\rho] \setminus i} \frac{X - \mathsf{pos}_j}{\mathsf{pos}_i - \mathsf{pos}_j} \cdot \Delta$$

$$f(X) := q(X) \prod_{i=1}^{\rho} (X - \mathsf{pos}_i) + \sum_{i=1}^{\rho} \tilde{\mathsf{s}}_i \cdot L_i(X)$$

return $\left\{\mathsf{s}^{(i)}\right\}_{i \in [\mathsf{m}]}$

$\mathsf{Reconstruct}(\{\mathsf{s}^{(i)}\}_{i \in \mathcal{S}})$

if $|\mathcal{S}| < \mathsf{t}$ **return** $\perp$

Parse $\mathcal{S} := \{i_1, \ldots, i_{\mathsf{t}}, \ldots\}$

for $k \in [\rho]$

for $j \in [\mathsf{t}]$

$$\Lambda_{i_j}(X) := \prod_{\zeta \in [\mathsf{t}] \setminus j} \frac{i_\zeta - X}{i_\zeta - i_j} \cdot (\Delta)$$

$$\mathsf{s}'_k := \sum_{j \in [t]} \Lambda_{i_j}(m + k) \cdot \mathsf{s}^{(j)}$$

return $\mathbf{s}' := (\mathsf{s}'_1, \ldots, \mathsf{s}'_\rho)$

Correctness. Observe that for all $i = 1, \ldots, \rho$, we have the following:

- $L_i(\mathsf{pos}_i) = \Delta$
- $L_j(\mathsf{pos}_i) = \Delta$ for all $j \in [\rho], j \neq i$
- $f(\mathsf{pos}_i) = \tilde{\mathsf{s}}_i \cdot \Delta = s_i \cdot \Delta^2$

Meanwhile, for $\lambda_{i_j}(X) := \prod_{\zeta \in [\mathsf{t}] \setminus [j]} \frac{i_\zeta - X}{i_\zeta - i_j}$, the polynomial we will be able to compute the polynomial $f(x) = \sum_{j \in [t]} \lambda_{i_j} \cdot s^{(j)}$ by correctness of Lagrange Interpolation. Consequently, $f(\mathsf{pos}_i)$ would return $\mathsf{s}_i \cdot \Delta^2$. However, we compute Λ_{i_j} instead, by multiplying with Δ to remove need for division. Consequently, the resulting polynomial has Δ multiplied throughout yielding a Δ^3 as the total offset.

Definition 3 (Vector of Sweeping Polynomials)**.** *Let* $\mathcal{C} \subset [\mathsf{m}]$ *such that* $|\mathcal{C}| = \mathsf{t} - \rho$*. Then, we have a vector of sweeping polynomials, denoted by* $\mathbf{sp}_{\mathcal{C}}(X) = (\mathsf{sp}_{1,\mathcal{C}}, \ldots, \mathsf{sp}_{\rho,\mathcal{C}})$ *where* $\mathsf{sp}_{i,\mathcal{C}}(X) := \sum_{j=0}^{\mathsf{t}-\rho} \mathsf{sp}_{i,j} \cdot X^j \in \mathbb{Z}[X]_{\leqslant \mathsf{t}-1}$ *is the unique polynomial whose degree is at most* $\mathsf{t} - 1$ *such that* $\mathsf{sp}_{i,\mathcal{C}}(\mathsf{m} + i) = \Delta^2$, $\mathsf{sp}_{i,\mathcal{C}}(\mathsf{m} + j) = 0$ *for* $j \in [\rho], j \neq i$, *and* $\mathsf{sp}_{i,\mathcal{C}}(j) = 0$ *for all* $j \in \mathcal{C}$*. Further, one can define* $\mathsf{sp}_{\max}$ *as the upper bound for the coefficients for the sweeping polynomials, i.e.,* $\mathsf{sp}_{max} := \left\{\mathsf{sp}_{i,j} | i \in \{1, \ldots, \rho\}, j \in \{0, \ldots, \mathsf{t}-1\}\right\}$

Lemma 6 (Existence of Sweeping Polynomial)**.** *For any* $\mathcal{C} \subset [\mathsf{m}]$ *with* $|\mathcal{C}| = \mathsf{t} - \rho$*, there exists* $\mathbf{sp}_{\mathcal{C}} \in (\mathbb{Z}[X]_{\leqslant \mathsf{t}-\rho})^\rho$ *satisfying Definition 3.*

Proof. For any $i = 1, \ldots, \rho$, we have that $\mathsf{sp}_{i,\mathcal{C}}(\mathsf{m}+i) = \Delta^2$ and $\mathsf{sp}_{i,\mathcal{C}}(j) = 0$ for $j \in \mathcal{C}$. Let $\mathcal{C} := (i_1, \ldots, i_{\mathsf{t}-\rho})$. In other words, we can use these evaluations to construct a polynomial as follows:

$$\mathsf{sp}_{i,\mathcal{C}}(X) := \Delta^2 \cdot \prod_{j=1}^{\mathsf{t}-\rho} \frac{(X - i_j)}{(\mathsf{m}+i) - i_j} \cdot \prod_{j \in [\rho] \setminus \{i\}} \frac{(X - (\mathsf{m}+j))}{(i-j)}$$

Note that $i_1, \ldots, i_j \in [\mathsf{m}]$ and are distinct. Therefore, $\prod_{j=1}^{\mathsf{t}-\rho} (\mathsf{m}+i) - i_j$ perfectly divides Δ and so does $\prod_{j \in [\rho] \setminus \{i\}} (i-j)$, which implies that the coefficients are all integers. Further, the degree of this polynomial is at most $\mathsf{t}-1$. Thus, $\mathsf{sp}_{i,\mathcal{C}}(X) \in \mathbb{Z}[X]_{\mathsf{t}-1}$. This defines the resulting vector of sweeping polynomials $\mathbf{sp}_{\mathcal{C}}$. □

Theorem 7. *Construction 9 is statistically private provided*

$$\ell_r \geqslant \ell_s + \lceil \log_2(\mathsf{sp}_{\max} \cdot (t-1) \cdot \rho) \rceil + 1$$

Proof. Let $\mathbf{s}, \mathbf{s}' \in [0, 2^{\ell_s})^\rho$ be two vectors of secrets. Then, $\tilde{\mathbf{s}} := \mathbf{s} \cdot \Delta$ and $\tilde{\mathbf{s}}' := \mathbf{s}' \cdot \Delta$. Let $\mathcal{C}$ denote an arbitrary subset of corrupted parties of size $|\mathcal{C}| = \mathsf{t} - \rho$. Further, let us assume that $\tilde{\mathbf{s}}$ is shared using the polynomial $f(X)$ as defined below:

$$f(X) := q(X) \cdot \prod_{k=1}^{\rho} (X - \mathsf{pos}_i) + \sum_{k=1}^{\rho} \tilde{\mathsf{s}}_i \cdot L_k(X)$$

where $L_k(X) := \prod_{j \in [\rho] \setminus \{k\}} \frac{X - \mathsf{pos}_j}{\mathsf{pos}_k - \mathsf{pos}_j} \cdot \Delta$. and $q(X)$ is a random polynomial of degree $\mathsf{t} - \rho - 1$.

Now observe that the adversary see $|\mathcal{C}| = \mathsf{t} - \rho$ shares corresponding to $f(i_j)$ for $i_j \in \mathcal{C}$. By Lagrange interpolation, this induces a one-to-one map from possible secrets to corresponding sharing polynomials. Specifically, we can use the vector of sweeping polynomials, as defined in Definition 3 to explicitly map any secret vector $\mathbf{s}^*$ to its sharing polynomial defined by $f(X) + \langle \mathbf{s}^* - \mathbf{s}, \mathbf{sp}_{\mathcal{C}}(X) \rangle$

In other words, the sharing polynomial to share $\mathbf{s}^*$ is defined by

$$f^*(X) = f(X) + \sum_{k=1}^{\rho} (\mathsf{s}_k^* - \mathsf{s}_k) \cdot \mathsf{sp}_{k,\mathcal{C}}(X)$$

One can verify the correctness. For example, to secret share s_1^*, at position $\mathsf{m}+1$, we get:

$$f^*(\mathsf{m}+1) = f(\mathsf{m}+1) + \sum_{k=1}^{\rho} (\mathsf{s}_k^* - \mathsf{s}_k) \cdot \mathsf{sp}_{k,\mathcal{C}}(X)$$

Now, observe that $f(\mathsf{m}+1) = \mathsf{s}_1 \cdot (\Delta^2)$. Meanwhile, $\mathsf{sp}_{1,\mathcal{C}}(\mathsf{m}+1) = \Delta^2$ while $\mathsf{sp}_{j,\mathcal{C}}(\mathsf{m}+1) = 0$ for $1 < j \leqslant \rho$. This simplifies to: $f^*(\mathsf{m}+1) = \mathsf{s}_1 \cdot \Delta^2 + (\mathsf{s}_1^* - \mathsf{s}_1) \cdot \Delta^2 = \mathsf{s}_1^* \cdot \Delta^2$. However, while we have an efficient mapping, note that $f^*(X)$ could have coefficients that are not of the prescribed form, i.e., coefficients do not lie in the range $[0, 2^{\ell_r + \kappa_s})$. We will call the event good if the coefficients lie in the range and bad even if one of the coefficients does not lie in the range.

Let us apply the above mapping to the secret $\mathbf{s}'$ and we have the resulting polynomial:

$$f'(X) = f(X) + \sum_{k=1}^{\rho} (\mathsf{s}'_k - \mathsf{s}_k) \cdot \mathsf{sp}_{k,\mathcal{C}}(X)$$

Now, observe that if $f'(X)$ was a good polynomial, then $f'(j) = f(j)$ for every $j \in \mathcal{C}$. It follows that if f' was good, then an adversary cannot distinguish whether the secret vector was $\mathbf{s}$ or $\mathbf{s}'$.

We will now upper bound the probability that f' was bad in at least one of the coefficients. We know that $|\mathsf{s}'_k - \mathsf{s}_k| \in [0, 2^{\ell_s})$ for $k = 1, \ldots, \rho$. Further all coefficients of $\mathsf{sp}_{k,\mathcal{C}}(X)$ are upper bounded by $\mathsf{sp}_{\max}$. Therefore, to any coefficient of $f(X)$, the maximum perturbation in value is: $2^{\ell_s} \cdot \mathsf{sp}_{\max} \cdot \rho$. Therefore, one requires that the original coefficients of f be sampled such that they lie in $[2^{\ell_s} \cdot \mathsf{sp}_{\max} \cdot \rho, 2^{\ell_r + \kappa_s} - 2^{\ell_s} \cdot \mathsf{sp}_{\max} \cdot \rho]$. In other words, the probability that one coefficient of f' is bad is:

$$\frac{2 \cdot 2^{\ell_s} \cdot \mathsf{sp}_{\max} \cdot \rho}{2^{\ell_r + \kappa_s}}$$

There are $\mathsf{t} - 1$ such coefficients. This gives us that the probability is $\leqslant 2^{-\kappa_s}$ assuming that $\ell_r \geqslant \ell_s + \lceil \log_2(\mathsf{sp}_{\max} \cdot (t-1) \cdot \rho) \rceil + 1$ □

A.2 Lattice-Based Assumptions

In this section, we will look at constructions based on three different lattice-based assumptions.

A.2.1 Learning with Rounding Assumption

We will begin by defining the learning with rounding (`LWR`) assumption, which can be viewed as a deterministic version of the learning with errors (`LWE`) assumption [80]. `LWR` was introduced by Banerjee *et al.* [9].

Definition 4 (Learning with Rounding). *Let $\rho, q, p \leftarrow\!\!\$\ \mathsf{LWRGen}(1^\rho)$ with $\rho, q, p \in \mathbb{N}$ such that $q > p$. Then, the Learning with Rounding assumption states that for all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathbf{s}, \mathbf{a}_0 \leftarrow\!\!\$\ \mathbb{Z}_q^\rho, \\ Y_0 := \lfloor \langle \mathbf{a}_0, \mathbf{s} \rangle \rfloor_p \\ Y_1 \leftarrow\!\!\$\ \mathbb{Z}_p, \mathbf{a}_1 \leftarrow\!\!\$\ \mathbb{Z}_q^\rho \\ b \leftarrow\!\!\$\ \{0,1\}, b' \leftarrow\!\!\$\ \mathcal{A}(\mathbf{a}_b, Y_b) \end{array} \right] = \frac{1}{2} + \text{negl}(\rho)$$

where $\lfloor x \rfloor_p = i$ where $i \cdot \lfloor q/p \rfloor$ is the largest multiple of $\lfloor q/p \rfloor$ that does not exceed x.

A.2.2 Learning with Errors Assumption

Definition 5 (Learning with Errors Assumption (LWE)). *Consider integers λ, L, q and a probability distribution χ on $\mathbb{Z}_q$, typically taken to be a normal distribution that has been discretized. Then, the $\mathtt{LWE}_{\lambda,L,q,\chi}$ assumption states that for all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{L\times\lambda}, \mathbf{x} \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda, \mathbf{e} \leftarrow\!\!\$\ \chi^L \\ \mathbf{y}_0 := \mathbf{A}\mathbf{x} + \mathbf{e} \\ \mathbf{y}_1 \leftarrow\!\!\$\ \mathbb{Z}_q^L, b \leftarrow\!\!\$\ \{0,1\}, b' \leftarrow\!\!\$\ \mathcal{A}(\mathbf{A}, \mathbf{y}_b) \end{array} \right] = \frac{1}{2} + \text{negl}(\rho)$$

Definition 6 (Hint-LWE [35, 45]). *Consider integers λ, L, q and a probability distribution χ' on $\mathbb{Z}_q$, typically taken to be a normal distribution that has been discretized. Then, the Hint-LWE assumption states that for all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{L\times\lambda}, \mathbf{k} \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda, \mathbf{e} \leftarrow\!\!\$\ \chi'^L \\ \mathbf{r} \leftarrow\!\!\$\ \mathbb{Z}_q^\lambda, \mathbf{f} \leftarrow\!\!\$\ \chi'^L \\ \mathbf{y}_0 := \mathbf{A}\mathbf{k} + \mathbf{e}, \mathbf{y}_1 \leftarrow\!\!\$\ \mathbb{Z}_q^L, b \leftarrow\!\!\$\ \{0,1\} \\ b' \leftarrow\!\!\$\ \mathcal{A}(\mathbf{A}, (\mathbf{y}_b, \mathbf{k} + \mathbf{r}, \mathbf{e} + \mathbf{f})) \end{array} \right] = \frac{1}{2} + \mathsf{negl}(\kappa)$$

where κ is the security parameter.

Kim *et al.* [59] demonstrates that the Hint-LWE assumption is computationally equivalent to the standard LWE assumption. In essence, this assumption posits that $\mathbf{y}_0$ maintains its pseudorandom properties from an adversary's perspective, even when provided with certain randomized information about the secret and error vectors. Consider a secure LWE instance defined by parameters (λ, m, q, χ), where χ represents a discrete Gaussian distribution with standard deviation σ. The corresponding Hint-LWE instance, characterized by (λ, m, q, χ'), where χ' denotes a discrete Gaussian distribution with standard deviation σ', remains secure under the condition $\sigma' = \sigma/\sqrt{2}$. As a result, we can decompose any $\mathbf{e} \in \chi$ into the sum $\mathbf{e}_1 + \mathbf{e}_2$, where both $\mathbf{e}_1$ and $\mathbf{e}_2$ are drawn from χ'.

A.3 Pseudorandom Functions

Definition 7 (Pseudorandom Function (PRF)). *A pseudorandom function family is defined by a tuple of PPT algorithms* $\mathsf{PRF} = (\mathsf{Gen}, \mathsf{Eval})$ *with the following definitions:*

- $\mathsf{pp}_{\mathsf{PRF}} \leftarrow\!\!\$\ \mathsf{Gen}(1^\kappa)$*: On input of the security parameter κ, the generation algorithm outputs the system parameters required to evaluate the function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ where $\mathcal{K}$ is the key space, $\mathcal{X}$ is the input space, and $\mathcal{Y}$ is the output space.*
- $y \leftarrow \mathsf{Eval}(\mathsf{k}, x)$*: On input of $x \in \mathcal{X}$ and a randomly chosen key $\mathsf{k} \leftarrow\!\!\$\ \mathcal{K}$, the algorithm outputs $y \in \mathcal{Y}$ corresponding to the evaluation of $F(k, x)$.*

We further require the following security property that: for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} b \leftarrow_\$ \{0,1\}, k \leftarrow_\$ \mathcal{K} \\ \mathcal{O}_0(\cdot) := F(k,\cdot), \mathcal{O}_1(\cdot) := U(\mathcal{Y}) \\ b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_b(\cdot)} \end{array} \right] \leqslant \frac{1}{2} + \mathrm{negl}(\kappa)$$

where $U(\mathcal{Y})$ outputs a randomly sampled element from $\mathcal{Y}$.

Definition 8 ((γ)-Key Homomorphic PRF)**.** *Let* PRF *be a pseudorandom function that realizes an efficiently computable function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ such that $(\mathcal{K}, \oplus)$ is a group. Then, we say that it is*

- *key homomorphic if: $(\mathcal{Y}, \otimes)$ is also a group and for every $\mathsf{k}_1, \mathsf{k}_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$ we get:* $\mathsf{Eval}(\mathsf{k}_1, x) \times \mathsf{Eval}(\mathsf{k}_2, x) = \mathsf{Eval}(\mathsf{k}_1 \oplus \mathsf{k}_2, x)$.
- *$\gamma = 1$-almost key homomorphic if: $\mathcal{Y} = \mathbb{Z}_p$ if for every $\mathsf{k}_1, \mathsf{k}_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$, there exists an error $e \in \{0,1\}$ we get:* $\mathsf{Eval}(\mathsf{k}_1, x) \times \mathsf{Eval}(\mathsf{k}_2, x) = \mathsf{Eval}(\mathsf{k}_1 \oplus \mathsf{k}_2, x) + e$.

A.4 Distributed Key Homomorphic PRF

Definition 9 (Distributed Key Homomorphic PRF (DPRF))**.** *A (t, m)-Distributed PRF is a tuple of PPT algorithms* $\mathsf{DPRF} := (\mathsf{Gen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{P\text{-}Eval}, \mathsf{Combine})$ *with the following syntax:*

- $\mathsf{pp}_{\mathsf{PRF}} \leftarrow_\$ \mathsf{Gen}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{m})$*: On input of the threshold t and number of parties m, and security parameter κ, the* Gen *algorithm produces the system parameter $\mathsf{pp}_{\mathsf{PRF}}$ which is impliclty consumed by all the other algorithms.*
- $\mathsf{k}^{(1)}, \ldots, \mathsf{k}^{(\mathsf{m})} \leftarrow_\$ \mathsf{Share}(\mathsf{k}, \mathsf{t}, \mathsf{r}, \mathsf{m})$*: On input of the number of parties m, corruption threshold t, reconstruction threshold r, and a key $\mathsf{k} \leftarrow_\$ \mathcal{K}$, the share algorithm produces the key share for each party.*
- $Y \leftarrow \mathsf{Eval}(\mathsf{k}, x)$*: On input of the PRF key k and input x, the algorithm outputs y corresponding to some pseudorandom function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$.*
- $y_i \leftarrow_\$ \mathsf{P\text{-}Eval}(\mathsf{k}^{(i)}, x)$*: On input of the PRF key share $\mathsf{k}^{(i)}$, the partial evaluation algorithm outputs a partial evaluation y_i on input $x \in \mathcal{X}$.*
- $Y' \leftarrow_\$ \mathsf{Combine}(\{y_i\}_{i \in \mathcal{S}})$*: On input of partial evaluations y_i corresponding to some subset of shares $\mathcal{S}$ such that $|\mathcal{S}| \geqslant \mathsf{t}$, the algorithm outputs Y'.*

We further require the following properties:

- *Correctness: We require that the following holds for any $\mathsf{m}, \mathsf{t}, \mathsf{r}, \kappa \in \mathbb{Z}$ with $\mathsf{t} < \mathsf{r} \leqslant \mathsf{m}$ and any set $\mathcal{S} \subseteq [m]$ with $|\mathcal{S}| \geqslant \mathsf{r}$, any input $x \in \mathcal{X}$:*

$$\Pr\left[Y = Y' \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{PRF}} \leftarrow_\$ \mathsf{Gen}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{r}, 1^\mathsf{m}), \mathsf{k} \leftarrow_\$ \mathcal{K} \\ \{\mathsf{k}^{(i)}\}_{i \in [\mathsf{m}]} \leftarrow_\$ \mathsf{Share}(\mathsf{k}), \{y_i \leftarrow \mathsf{P\text{-}Eval}(\mathsf{k}^{(i)}, x)\}_{i \in \mathcal{S}} \\ Y' \leftarrow \mathsf{Combine}(\{y_i\}_{i \in \mathcal{S}}), Y = \mathsf{Eval}(\mathsf{k}, x) \end{array} \right] = 1$$

- *Pseudorandomness with Static Corruptions: We require that for any integers $\mathsf{t}, \mathsf{r}, \mathsf{m}$ with $\mathsf{t} < \mathsf{r} \leqslant \mathsf{m}$, and for all PPT adversary $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr\left[\begin{array}{c} b = b' \\ |\mathbf{K} \cup \{j : (j, x^*) \in \mathbf{E}\}| \leqslant \mathsf{t} \\ \\ \\ \\ \\ \end{array} \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{PRF}} \leftarrow_\$ \mathsf{Gen}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{r}, 1^\mathsf{m}), \mathsf{k} \leftarrow_\$ \mathcal{K} \\ (st, \mathbf{K}) \leftarrow_\$ \mathcal{A}(\mathsf{pp}_{\mathsf{PRF}}) \\ \{\mathsf{k}^{(i)}\}_{i \in [\mathsf{m}]} \leftarrow_\$ \mathsf{Share}(\mathsf{k}, \mathsf{t}, \mathsf{r}, \mathsf{m},) \\ (st, x^*) \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{Eval}}}(\{\mathsf{k}^{(i)}\}_{i \in \mathbf{K}}) \\ b \leftarrow_\$ \{0,1\}, Y_0 \leftarrow_\$ \mathsf{Eval}(\mathsf{k}, x^*) \\ Y_1 \leftarrow_\$ \mathcal{U}(\mathcal{Y}), b' \leftarrow_\$ \mathcal{A}^{\mathcal{O}_{\mathsf{Eval}}}(st, Y_b) \end{array} \right] = 1$$

  where:

$$\begin{array}{l} \underline{\mathcal{O}_{\mathsf{Eval}}(i, x)\qquad\qquad} \\ \mathbf{E} := \mathbf{E} \cup \{(i, x)\} \\ \textbf{return } \mathsf{P\text{-}Eval}(\mathsf{k}^{(i)}, x) \end{array}$$

- *Key Homomorphic: We require that if* $(\mathcal{K}, \oplus), (\mathcal{Y}, \otimes)$ *are groups such that:*
  - $\forall\ x \in \mathcal{X}, \forall\ \mathsf{k}_1, \mathsf{k}_2 \in \mathcal{K}, \mathsf{Eval}(\mathsf{k}_1, x) \otimes \mathsf{Eval}(\mathsf{k}_2, x) = \mathsf{Eval}(\mathsf{k}_1 \oplus \mathsf{k}_2, x)$*, and*
  - $\forall\ x \in \mathcal{X}\ ,\ \forall\ \mathsf{k}_1, \mathsf{k}_2 \in \mathcal{K}, \left\{\mathsf{k}_b^{(j)} \leftarrow_\$ \mathsf{Share}(\mathsf{k}_b)\right\}_{j \in [m], b \in \{1,2\}}$,

    $\forall\ j \in [m], y_{1,2}^{(j)} := \left(\mathsf{P\text{-}Eval}(\mathsf{k}_1^{(j)}, x) \otimes \mathsf{P\text{-}Eval}(\mathsf{k}_2^{(j)}, x)\right)$*, and* $\forall \mathcal{S} \subseteq [m]$ *with* $|\mathcal{S}| \geqslant \mathsf{r}$*,* $\mathsf{Combine}\left(\left\{y_{1,2}^{(j)}\right\}_{j \in \mathcal{S}}\right) = \mathsf{Eval}(\mathsf{k}_1 \oplus \mathsf{k}_2, x)$

A.5 Class Groups Framework

Class group-based cryptography is a cryptographic technique that originated in the late 1980s, with the idea that the class group of ideals of maximal orders of imaginary quadratic fields may be more secure than the multiplicative group of finite fields [23, 70]. The CL framework was first introduced by the work of Castagnos and Laguillaumie [32]. This framework operates on a group where there exists a subgroup with support for efficient discrete logarithm construction. Subsequent works [33, 29, 30, 89, 21] have refined the original framework. The framework has been used in various applications over the years [33, 29, 30, 93, 44, 50, 88, 21, 58]. Meanwhile, class group cryptography itself has been employed in numerous applications [91, 92, 64, 18, 31, 34, 60, 24, 7, 1, 42, 41, 8].

Broadly, the framework is defined by two functions - $\mathsf{CLGen}, \mathsf{CLSolve}$ with the former outputting a tuple of public parameters. The elements of this framework are the following:

- *Input Parameters*: κ_c is the computational security parameter, κ_s is a statistical security parameter, a prime p such that $p > 2^{\kappa_c}$, and uniform randomness ρ that is used by the CLGen algorithm and is made public.
- *Groups*: $\widehat{\mathbb{G}}$ is a finite multiplicative abelian group, $\mathbb{G}$ is a cyclic subgroup of $\widehat{\mathbb{G}}$, $\mathbb{F}$ is a subgroup of $\mathbb{G}$, $H = \{x^p, x \in \mathbb{G}\}$
- *Orders*: $\mathbb{F}$ has order p, $\widehat{\mathbb{G}}$ has order $p \cdot \hat{s}$, $\mathbb{G}$ has order $p \cdot s$ such that s divides $\hat{s}$ and $gcd(p, \hat{s}) = 1, gcd(p, s) = 1$, H has order s and therefore $\mathbb{G} = \mathbb{F} \times H$.
- *Generators*: f is the generator of $\mathbb{F}$, g is the generator of $\mathbb{G}$, and h is the generator of H with the property that $g = f \cdot h$
- *Upper Bound*: Only an upper bound $\bar{s}$ of $\hat{s}$ (and s) is provided.
- *Additional Properties*: Only encodings of $\widehat{\mathbb{G}}$ can be recognized as valid encodings and $s, \hat{s}$ are unknown.
- *Distributions*: $\mathcal{D}$ (resp. $\mathcal{D}_p$) be a distribution over the set of integers such that the distribution $\{g^x : x \leftarrow_\$ \mathcal{D}\}$ (resp. $\{g_p^x : x \leftarrow_\$ \mathcal{D}_p\}$) is at most distance $2^{-\kappa_s}$ from the uniform distribution over $\mathbb{G}$ (resp. H).
- *Additional Group and its properties*: $\widehat{\mathbb{G}}^p = \left\{x^p, x \in \widehat{\mathbb{G}}\right\}$, $\widehat{\mathbb{G}}$ factors as $\widehat{\mathbb{G}}^p \times \mathbb{F}$.[4] Let $\bar{\omega}$ be the group exponent of $\widehat{\mathbb{G}}^p$. Then, the order of any $x \in \widehat{\mathbb{G}}^p$ divides $\bar{\omega}$.[5]

Remark 4. The motivations behind these additional distributions are as follows. One can efficiently recognize valid encodings of elements in $\widehat{\mathbb{G}}$ but not $\mathbb{G}$. Therefore, a malicious adversary $\mathcal{A}$ can run our constructions by inputting elements belonging to $\widehat{\mathbb{G}}^p$ (rather than in H). Unfortunately, this malicious behavior cannot be detected which allows $\mathcal{A}$ to obtain information on the sampled exponents modulo $\bar{\omega}$ (the group exponent of $\widehat{\mathbb{G}}^p$). By requiring the statistical closeness of the induced distribution to uniform in the aforementioned groups allows flexibility in proofs. Note that the assumptions do not depend on the choice of these two distributions. Further, the order s of H and group exponent $\bar{\omega}$ of $\widehat{\mathbb{G}}^p$ are unknown and the upper bound $\bar{s}$ is used to instantiate the aforementioned distribution. Specifically, looking ahead we will set $\mathcal{D}_H$ to be the uniform distribution over the set of integers $[B]$ where $B = 2^{\kappa_s} \cdot \bar{s}$. Using Lemma 8, we get that the distribution is

[4] Recall that p and $\hat{s}$ are co-prime.

[5] This follows from the property that the exponent of a finite Abelian group is the least common multiple of its elements.

less than $2^{-\kappa_s}$ away from uniform distribution in H. In our constructions we will set $\kappa_s = 40$. We will make this sampling more efficient for our later constructions. We refer the readers to Tucker [89, §3.1.3, §3.7] for more discussions about this instantiation. Finally, as stated we will also set $\widehat{\mathcal{D}} = \mathcal{D}$ and $\widehat{\mathcal{D}}_H = \mathcal{D}_H$.

We also have the following lemma from Castagnos, Imbert, and Laguillaumie [31] which defines how to sample from a discrete Gaussian distribution.

Lemma 8. *Let $\mathbb{G}$ be a cyclic group of order n, generated by g. Consider the random variable X sampled uniformly from $\mathbb{G}$; as such it satisfies $\Pr[X = h] = \frac{1}{n}$ for all $h \in \mathbb{G}$. Now consider the random variable Y with values in $\mathbb{G}$ as follows: draw y from the discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z},\sigma}$ with $\sigma \geqslant n\sqrt{\frac{\ln(2(1+1/\epsilon))}{\pi}}$ and set $Y := g^y$. Then, it holds that:*

$$\Delta(X, Y) \leqslant 2\epsilon$$

Remark 5. By definition, the distribution $\{g^x : x \leftarrow_\$ \mathcal{D}\}$ is statistically indistinguishable from $\{g^y : y \leftarrow_\$ \{0, \ldots, p \cdot s - 1\}\}$. Therefore, it follows that $\{x \mod p \cdot s : x \leftarrow_\$ \mathcal{D}\}$ is statistically indistinguishable from $\{x : x \leftarrow_\$ \{0, \ldots, p \cdot s - 1\}\}$. Similarly, $\{x \mod s : x \leftarrow_\$ \mathcal{D}_p\}$ is statistically indistinguishable from $\{x : x \leftarrow_\$ \{0, \ldots, s - 1\}\}$. Furthermore, sampling a value x corresponding to $\mathcal{D}$ is statistically indistinguishable from the uniform distribution in $\{0, \ldots, s - 1\}$ because s divides $p \cdot s$.

Definition 10 (Class Group Framework)**.** *The framework is defined by two algorithms* $(\mathsf{CLGen}, \mathsf{CLSolve})$ *such that:*

- $\mathsf{pp}_{\mathsf{CL}} = (p, \kappa_c, \kappa_s, \bar{s}, f, h, \widehat{\mathbb{G}}, \mathbb{F}.\mathcal{D}, \mathcal{D}_H, \widehat{\mathcal{D}}, \widehat{\mathcal{D}}_H, \rho) \leftarrow_\$ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s}, p; \rho)$
- *The* DL *problem is easy in $\mathbb{F}$, i.e., there exists a deterministic polynomial algorithm* $\mathsf{CLSolve}$ *that solves the discrete logarithm problem in $\mathbb{F}$:*

$$\Pr\left[x = x' \;\middle|\; \begin{array}{l} \mathsf{pp}_{\mathsf{CL}} = \leftarrow_\$ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s}, p; \rho) \\ x \leftarrow_\$ \mathbb{Z}/p\mathbb{Z}, X = f^x; \\ x' \leftarrow \mathsf{CLSolve}(\mathsf{pp}_{\mathsf{CL}}, X) \end{array} \right] = 1$$

Definition 11 (Hard Subgroup Membership Assumption ($\mathtt{HSM}_{\mathsf{M}}$) Assumption [33])**.** *Let κ be a the security parameter with prime p such that $|p| \geqslant \kappa$. Let* $(\mathsf{CLGen}, \mathsf{CLSolve})$ *be the group generator algorithms as defined in Definition 10, then the $\mathtt{HSM}_{\mathsf{M}}$ assumption requires that that $\mathtt{HSM}_{\mathsf{M}}$ problem be hard in $\mathbb{G}$ even with access to the* Solve *algorithm. More formally, let $\mathcal{D}$ (resp. $\mathcal{D}_p$) be a distribution over the set of integers such that the distribution $\{g^x : x \leftarrow_\$ \mathcal{D}\}$ (resp. $\{g_p^x : x \leftarrow_\$ \mathcal{D}_p\}$) is at most distance $2^{-\kappa}$ from the uniform distribution over $\mathbb{G}$ (resp. H). Then, we say that the $\mathtt{HSM}_{\mathsf{M}}$ problem is hard if for all PPT adversaries $\mathcal{A}$, there exists a negligible function* $\operatorname{negl}(\kappa)$ *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{CL}} \leftarrow_\$ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s}, p; \rho) \\ x \leftarrow_\$ \mathcal{D}, x' \leftarrow_\$ \mathcal{D}_p \\ b \leftarrow_\$ \{0,1\}; Z_0 = g^x; Z_1 = g_p^{x'} \\ b' \leftarrow_\$ \mathcal{A}^{\mathsf{Solve}(\mathsf{pp}_{\mathsf{CL}}, \cdot)}(\mathsf{pp}_{\mathsf{CL}}, Z_b) \end{array} \right] \leqslant \frac{1}{2} + \operatorname{negl}(\kappa)$$

When dealing with groups of known order, one can sample elements in a group $\mathbb{G}$ easily by merely sampling exponents modulo the group order and then raising the generator of the group to that exponent. Unfortunately, note that here neither the order of $\mathbb{G}$ (i.e., ps) nor that of H (i.e. s) is known. Therefore, we instead use the knowledge of the upper-bound $\bar{s}$ of s to instantiate the distributions $\mathcal{D}$ and $\mathcal{D}_p$ respectively. This choice of choosing from the distributions $\mathcal{D}$ and $\mathcal{D}_p$ respectively allows for flexibility of various proofs.

B Naive Construction of OPA

We present a naive construction that achieves our desired one-shot behavior. This construction is built from Shamir's Secret Sharing algorithm, as recalled in Construction 6. This shall serve as the baseline construction.

- Setup: Simply runs the setup algorithm for the secret sharing algorithm SS
- Enc: Client i, with inputs x_i secret shares the input, for each element in $\mathbf{x}_i$. In other words, for $i = 1, \ldots, L$, do $\left\{x_i^{(j)}\right\}_{j \in [m]} \leftarrow_\$ \mathsf{SS.Share}(x_i, \mathsf{t}, \mathsf{r}, \mathsf{m})$

It sets $\mathsf{k}_i := \emptyset, \mathbf{ct}_i := \emptyset$ and $\mathsf{aux}_i^{(j)} := \left\{x_i^{(j)}\right\}_{i=1}^{L}$

- $\mathsf{C\text{-}Combine}$: Committee member j, simply adds up all the shares it has received, element-by-element.

  $\mathsf{AUX}^{(j)} \sum_{i=1}^{n} \mathsf{aux}_i^{(j)}$

- $\mathsf{S\text{-}Combine}$: Upon receiving from at least r committee members, it runs $\mathsf{SS.Reconstruct}(\left\{\mathsf{AUX}^{(j)}\right\}_{j\in\mathcal{S}})$, element-by-element, to get the sum of vectors.

- $\mathsf{Aggregate}$: The previous algorithm already gives the sum.

This algorithm has the following communication:

- Client i to Committee: L field elements per committee member for a total of $\mathsf{m}L$ field elements.
- Client i to Server: 0
- Committee j to Server: L field elements.
- Total Sent/Received per Client: $\mathsf{m}L$ field elements.
- Total Sent/Received by Server: $\mathsf{m}L$ field elements.
- Total Sent/Received per Committee Member: $L + nL$ field elements.

Remark 6. It is important to make a few observations. This solution resembles a multi-server solution where the clients simply secret-share the inputs with each server, and the servers collaborate to recover the sum. However, this observation also implies that the role of the committee members, while running a deterministic procedure, is proportional to the length of the input. This is undesirable as these committee members are simply other clients and such a heavy computation is undesirable.

It is also to be noted that even a simple solution such as using a one-time pad to mask the input, and then secret share the mask suffers from the same bottleneck as the mask has to be at least the length of the input.

Computation Performance of $\mathsf{OPA}_{\mathtt{LWR}}$ vs Naive Solution. We now list the computation costs that were measured. Note that these experiments were done for $L = 1$. The motivation behind these experiments was to study how the two constructions fared across the computation time incurred by the Naive Solution and $\mathsf{OPA}_{\mathtt{LWR}}$, without any optimizations including packed secret sharing.

Our experiments indicate that, in addition to the significant savings in communication cost, $\mathsf{OPA}_{\mathtt{LWR}}$ outperforms the Naive Solution when it comes to the performance of the committee. While the the role of the committee is the same in both constructions, the cost of the committee for $\mathsf{OPA}_{\mathtt{LWR}}$ does not scale linearly with the input, while the naive solution incurs a linear growth cost. Further, note that currently, $\mathsf{OPA}_{\mathtt{LWR}}$ requires the committee to add shares of 2^{11} elements. It is clear when aggregating vectors of length $> 2^{11}$, $\mathsf{OPA}_{\mathtt{LWR}}$ will outperform the naive solution. Meanwhile, it seems clear that $\mathsf{OPA}_{\mathtt{LWR}}$'s client cost which involves secret sharing the seed and then masking a vector of length L will be slower than simply secret sharing a vector of length L. Similarly, the cost of simply reconstructing the summed up vectors of length for the server in the naive solution will be better than the multi-step process of $\mathsf{OPA}_{\mathtt{LWR}}$ which includes reconstructing the seed, then evaluating the said seed, before finally unmasking. For completeness, we plot the server and client costs comparison in Figure 7. Note that we have scaled the costs, in the case of the naive solution, by 100 (i.e., if the actual cost was 1 unit, we plotted it as 100 units).

C Constructions in CL Framework

Construction 10 (PRF in CL Framework)**.** Let $(\mathsf{CLGen}, \mathsf{CLSolve})$ be the class group framework as defined in Definition 10. Then, let $\mathsf{pp}_{\mathsf{CL}} \leftarrow\$ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s})$. Further, let $\mathtt{H} : \mathcal{X} \to \mathbb{H}, \mathtt{H}' : \mathcal{X} \to \mathcal{K}$ be a hash function. Then, consider the following definition of $\mathcal{K} = \mathcal{D}_H, \mathcal{X} = \{0,1\}^*, \mathcal{Y} = \mathbb{H}$, $F_{\mathsf{CL}}(k, x) = \mathtt{H}(x)^{k \cdot \mathtt{H}'(x,1)}$.

Protocol Naive Construction of OPA

One-Time System Parameters Generation

Run $\mathsf{pp} \leftarrow\$ \mathsf{SS.Setup}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{r}, 1^\mathsf{m})$
Output Committee of size m.

Data Encryption Phase by Client i in iteration ℓ

Let $\mathbf{x}_{i,\ell}$ be the input of Client i in iteration ℓ.
for $in = 1, \ldots, L$ **do**
$\quad \mathsf{ct}_{i,\ell} = \perp, \left\{\mathsf{aux}_{i,\ell}^{(j)}[in]\right\}_{j \in [\mathsf{m}]} \leftarrow \mathsf{SS.Share}(x_{i,\ell}[in], \mathsf{t}, \mathsf{m}, \ell)$
Send $\mathbf{ct}_{i,\ell}$ to the Server
Send $\mathsf{aux}_{i,\ell}^{(j)}$ to committee member j for each $j \in [\mathsf{m}]$.

Set Identification Phase by Committee Member j in iteration ℓ

Let $\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}$ be the inputs received by Committee member j from Clients $i \in \mathcal{C}^{(j)}$
Send $\mathcal{C}^{(j)}$ to server // This can be a bit string of length n where i-th bit is 1 if $i \in \mathcal{C}^{(j)}$

Set Intersection Phase by Server in iteration ℓ

Let $\left\{\mathcal{C}^{(j)}\right\}_{j \in \mathcal{S}}$ be the client sets received from committee members $j \in \mathcal{S}$.
assert $|\mathcal{S}| \geqslant \mathsf{r}$
Compute $\mathcal{C} := \cap_{j \in \mathcal{S}} \mathcal{C}^{(j)}$ // This is bit-wise AND operation of the r bit strings.
Send $\mathcal{C}$ to committee members in $\mathcal{S}$

Data Combination Phase by Committee Member j in iteration ℓ

Let $\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}$ be the inputs received by Committee member j from Clients $i \in \mathcal{C}$
for $in = 1, \ldots, L$ **do**
$\quad$ Compute $\mathsf{AUX}^{(j)}[in] \leftarrow \sum_{i \in \mathcal{C}} \mathsf{aux}_{i,\ell}^{(j)}[in]$
Send $\mathsf{AUX}^{(j)}$ to server

Data Aggregation Phase by Server in iteration ℓ

for $in = 1, \ldots, L$ **do**
$\quad$ Let $\left\{\mathsf{AUX}_\ell^{(j)}[in]\right\}_{j \in \mathcal{S}}$ be the inputs received by the server with $|\mathcal{S}| \geqslant \mathsf{t}$.
$\quad$ Run $X_\ell[in] \leftarrow \mathsf{SS.Reconstruct}(\left\{\mathsf{AUX}_\ell^{(j)}[in]\right\}_{j \in \mathcal{S}})$

Figure 6: Our Naive Construction of OPA using just a secret sharing scheme.

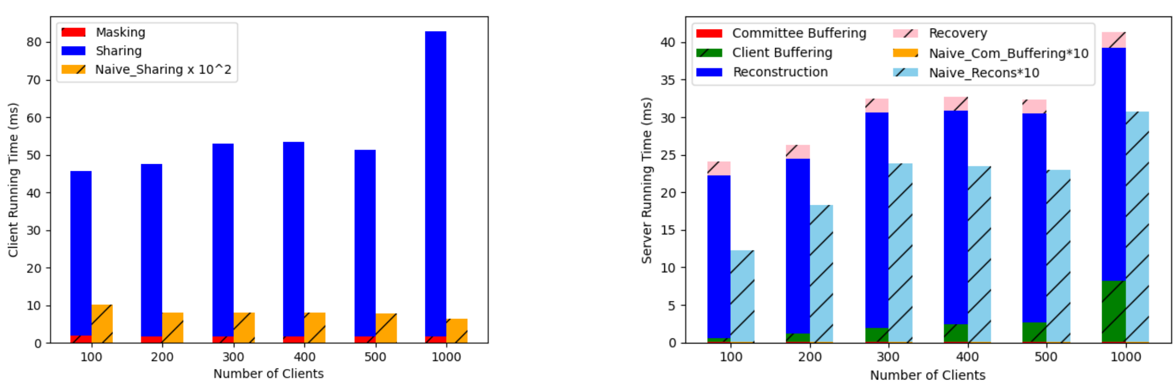


Figure 7: Comparison of Running Times between $\mathsf{OPA}_{\mathtt{LWR}}$ and the Naive Solution. Values that scale linearly with n are marked with a slash on the entries.

Remark 7. Note that the order of $\mathbb{H}$ is unknown. Therefore, one has to rely on $\mathcal{D}_H$ to hash into $\mathbb{H}$. Most

Protocol Distributed PRF

$\underline{\mathsf{Gen}(1^{\kappa}, 1^{\mathsf{r}}, 1^{\mathsf{m}})}$

Parse $\kappa = (\kappa_s, \kappa_c)$
Run $\mathsf{pp}_{\mathsf{CL}} \leftarrow\$ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s})$
Set $\ell_s := |\mathsf{M}|$
Sample $\mathtt{H} : \mathcal{X} \to \mathbb{H}$
return $\mathsf{pp}_{\mathsf{PRF}} := (\mathsf{pp}_{\mathsf{CL}}, \mathsf{LISS}.\mathsf{pp}_{\mathsf{SS}}, \mathtt{H})$

$\underline{\mathsf{Share}(\mathsf{k}, \mathsf{r}, \mathsf{m})}$

Run $\mathsf{k}^{(1)}, \ldots, \mathsf{k}^{(\mathsf{m})} \leftarrow\$ \mathsf{LISS}.\mathsf{Share}(\mathsf{k}, \mathsf{r}, \mathsf{m})$
return $\mathsf{k}^{(1)}, \ldots, \mathsf{k}^{(\mathsf{m})}$

$\underline{\mathsf{Eval}(\mathsf{k}, x)}$

Compute $Y = \mathtt{H}(x)^{\Delta^3 \cdot \mathsf{k}}$
return Y

$\underline{\mathsf{P\text{-}Eval}(\mathsf{k}^{(i)}, x)}$

Compute $y_i = \mathtt{H}(x)^{\Delta \cdot \mathsf{k}^{(i)}}$
return y_i

$\underline{\mathsf{Combine}(\{y_i\}_{i \in \mathcal{S}})}$

Run $\Lambda_{i \in \mathcal{S}} = \mathsf{LISS}.\mathsf{GetCoeff}(\mathcal{S})$
Compute $Y' = \prod_{i \in \mathcal{S}} y_i^{\Lambda_i}$
return Y'

Figure 8: Construction of Distributed PRF based on the $\mathsf{LISS} := (\mathsf{Share}, \mathsf{GetCoeff}, \mathsf{Reconstruct})$ scheme of [21], with $\mathsf{pp}_{\mathsf{SS}}$ denoting the public parameters of the LISS scheme. Recall that the offset $\Delta := \mathsf{m}!$ where m is the number of shares generated.

recently, [81] showed how to hash into groups of unknown order to ensure that discrete logarithm is unknown. However, for our applications, this is not a concern. Indeed, one can simply compute the hash function as $h^{H'(x)}$ where $H'(x)$ hashed in $\mathcal{D}_H$.

Theorem 9. *Construction 10 is a secure PRF where* $\mathtt{H}$ *is modeled as a random oracle under the* $\mathtt{HSM}_{\mathsf{M}}$ *assumption.*

The proof is deferred to Section F.1

C.1 Distributed PRF in CL Framework

We build our construction of Distributed PRF from the Linear Secret Sharing Scheme $\mathsf{LISS} := (\mathsf{Share}, \mathsf{GetCoeff}, \mathsf{Reconstruct})$ with $\mathsf{pp}_{\mathsf{SS}}$ denoting the public parameters of the LISS scheme. We specifically employ the Shamir Secret Sharing scheme over the Integers, as defined in [21].

Construction 11 (Distributed PRF in CL Framework)**.** A (r, m)-Distributed PRF is a tuple of PPT algorithms $\mathsf{DPRF} := (\mathsf{Gen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{P\text{-}Eval}, \mathsf{Combine})$ with the algorithms as defined in Figure 8. For simplicity, in the construction below we will set the corruption threshold $\mathsf{t} = \mathsf{r} - 1$. Though, the construction also holds for a lower t.

Theorem 10. *In the Random Oracle Model, if Construction 10 is a secure pseudorandom function if Integer Secret Sharing is statistically private, then Construction 11 is pseudorandom in the static corruptions setting.*

The proof of security and correctness can be found in Section F.2.

C.2 Construction of One-shot Private Aggregation without Leakage Simulation in CL Framework

Construction 12. We present the construction of One-shot Private Aggregation without Leakage Simulation, in the CL Framework in Figure 9.

D Construction from LWR

D.1 Distributed PRF from LWR

Let us revisit Construction 14. First, observe that the key space is from $\mathbb{Z}_q^{\rho}$ which implies that the order of $\mathcal{K}$ is known. Further, the computation occurs over a group whose structure and order is known. This is a departure from the construction based on the $\mathtt{HSM}_{\mathsf{M}}$ assumption. Consequently, by assuming that both p and q are primes, one can avoid integer secret sharing but instead rely on traditional Shamir's Secret Sharing over a field, which we defined earlier (see Construction 6).

Protocol OPA'

System Parameters Generation

Run $\mathsf{pp} \leftarrow\$ \mathsf{CL.Gen}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{r}, 1^\mathsf{m})$

Server Initiating Iteration ℓ

Select n clients $\mathcal{C}_\ell$.
Select $\mathsf{m} = \lfloor \log_2 n \rfloor$-sized committee
Broadcast to n clients the committee.

Data Encryption Phase by Client i in iteration ℓ

Input: $\mathbf{x}_{\mathbf{i},\ell} = (x_{i,\ell}^{(1)}, \dots, x_{i,\ell}^{(L)})$
Sample $\mathsf{k}_{i,1}, \dots, \mathsf{k}_{i,L} \leftarrow\$ \mathsf{DPRF}.\mathcal{K}$
for $in = 1, \dots, L$ **do**
  $h_{i,\ell}^{(in)} = \mathsf{DPRF.Eval}(\mathsf{k}_{i,in}, \ell)$
  Compute $\mathsf{ct}_{i,\ell}^{(in)} = f^{x_{i,\ell}^{(in)}} \cdot h_{i,\ell}^{(in)}$
  Compute $\left\{\mathsf{k}_{i,in}^{(j)}\right\}_{j \in [\mathsf{m}]} \leftarrow\$ \mathsf{DPRF.Share}(\mathsf{k}_{i,in}, \mathsf{t}, \mathsf{r}, \mathsf{m})$
  For $j \in [\mathsf{m}]$, set $\left\{\mathsf{aux}_{i,\ell}^{(j,in)}\right\} = \mathsf{DPRF.P\text{-}Eval}(\mathsf{k}_{i,in}^{(j)}, \ell$
Send $\mathsf{ct}_{i,\ell}^{(1)}, \dots, \mathsf{ct}_{i,\ell}^{(L)}$ to the Server
Send $\mathsf{aux}_{i,\ell}^{(j,1)}, \dots, \mathsf{aux}_{i,\ell}^{(j,L)}$ to committee member j $\forall j \in [\mathsf{m}]$, via Server appropriately encrypted

Set Intersection Phase by Server in iteration ℓ

For $j \in [\mathsf{m}]$, let $\mathcal{C}^{(j)} := \left\{i : c_{i,\ell}^{(j)} \text{was received by server}\right\}$
Let $C^{(0)} := \{i : \mathbf{ct}_{i,\ell} \text{was received by the server}\}$
Compute $\mathcal{C} := \cap_{j \in \mathcal{S} \cup \{0\}} \mathcal{C}^{(j)}$ // This is bit-wise AND operation of the $\mathsf{r}+1$ bit strings.
assert $|\mathcal{C}| \geqslant (1-\delta)n$
Send $\mathcal{C}, \left\{c_{i,\ell}^{(j)}\right\}_{i \in \mathcal{C}}$ for every j in $[\mathsf{m}]$

Data Combination Phase by Committee Member j in iteration ℓ

Input: $\left\{\mathsf{aux}_{i,\ell}^{(j,1)}, \dots, \mathsf{aux}_{i,\ell}^{(j,L)}\right\}_{i \in \mathcal{C}}$
for $in = 1, \dots, L$ **do**
  $(\mathsf{AUX}^{(j,in)}) \leftarrow \otimes_{i \in \mathcal{C}}(\mathsf{aux}_{i,\ell}^{(j,in)})$
Send $\mathsf{AUX}^{(j,1)}, \dots, \mathsf{AUX}^{(j,L)}$ to Server

Data Aggregation Phase by Server in iteration ℓ

Input: $\left\{\left\{\mathsf{AUX}_\ell^{(j,in)}\right\}_{j \in \mathcal{S}}, \left\{\mathsf{ct}_{i,\ell}^{(in)}\right\}_{i \in \mathcal{C}}\right\}_{in \in [L]}, |\mathcal{S}| \geqslant \mathsf{t}.$
for $in = 1, \dots, L$ **do**
  Run $\mathsf{AUX}_\ell^{(in)} \leftarrow \mathsf{DPRF.Combine}(\left\{\mathsf{AUX}_\ell^{(j,in)}\right\}_{j \in \mathcal{S}})$
  Run $X_\ell^{(in)} \leftarrow \mathsf{CLSolve}(\mathsf{pp}, (\mathsf{AUX}_\ell^{(in)})^{-1} \cdot \prod_{i \in \mathcal{C}} \mathsf{ct}_{i,\ell}^{(in)})$
return $\left\{X_\ell^{(in)}\right\}_{in \in [L]}$

Figure 9: In this figure, we present the modified steps for OPA'. For simplicity, we present only the semi-honest construction. For malicious security, we augment similarly with a second mask.

We saw earlier that Construction 14 was only almost key homomorphic, i.e.:

$$F(\mathbf{k}_1 + \mathbf{k}_2, x) = F(\mathbf{k}_1, x) + F(\mathbf{k}_2, x) + e$$

where $e \in \{0, 1\}$. It also follows that:

$$T \cdot F(\mathbf{k}_1, x) = F(T \cdot \mathbf{k}_0, x) - e_T$$

where $e_T \in 0, \dots, T$. This becomes a cause for concern as, in the threshold construction using the Shamir Secret Sharing over the field as shown in Construction 6, one often recombines by multiplying with a Lagrange coefficient λ_{i_j}. Unfortunately, multiplying the result by λ_{i_j} implies that the error term $e_{\lambda_{i_j}} \in \{0, \dots, i_j\}$. The requirement is that this error term should not become "too large". However, interpreting Lagrange coefficients as elements in $\mathbb{Z}_p$ results in the error term failing to be low-norm leading to error propagation. To mitigate this, we use techniques quite similar to Construction 8 by essentially clearing the denominator by multiplying with $\Delta := \mathsf{m}!$. This is a technique made popular by the work of Shoup [84] and later used in several other works including in the context of lattice-based cryptography by Agrawal *et al.* [4] and later to construct a distributed key homomorphic PRF from any almost key homomorphic PRF by Boneh *et al.* [19].

Protocol Distributed PRF from Learning with Rounding Assumption

$\underline{\mathsf{Gen}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{m})}$

Parse $\kappa = (\rho)$
Run $\mathsf{pp}_{\mathsf{LWR}} = (\rho, q, p) \leftarrow\$ \mathsf{LWRGen}(1^\rho)$
Set $\ell_s := |q|$
Set u such that $\lfloor p/u \rfloor > (\Delta + 1)\mathsf{t}\Delta$
Set v such that $\lfloor u/v \rfloor > \Delta\mathsf{t}$
Sample $\mathtt{H} : \mathcal{X} \to \mathbb{Z}_q^\rho$
return $\mathsf{pp}_{\mathsf{PRF}} := (\mathsf{pp}_{\mathsf{LWR}}, \mathsf{pp}_{\mathsf{SS}}, \mathtt{H}, u)$

$\underline{\mathsf{Share}(\mathbf{k} \in \mathbb{Z}_q^\rho, \mathsf{t}, \mathsf{m})}$

for $i = 1, \ldots, \rho$ **do**
  $\mathsf{k}_i^{(1)}, \ldots, \mathsf{k}_i^{(\mathsf{m})} \leftarrow\$ \mathsf{SS.SecretShare}(\mathsf{k}_i, \mathsf{t}, \mathsf{m})$
return $\left\{\mathbf{k}^{(j)} = (\mathsf{k}_1^{(j)}, \ldots, \mathsf{k}_\rho^{(j)})\right\}_{j \in [\mathsf{m}]}$

$\underline{\mathsf{Eval}(\mathbf{k}, x)}$

Compute $Y = \left\lfloor \Delta \left\lfloor \Delta \lfloor \langle \mathtt{H}(x), \mathbf{k} \rangle \rfloor_p \right\rfloor_u \right\rfloor_v$
return Y

$\underline{\mathsf{P\text{-}Eval}(\mathbf{k}^{(i)}, x)}$

Compute $y_i = \left\lfloor \Delta \left\lfloor \langle \mathtt{H}(x), \mathbf{k}^{(i)} \rangle \right\rfloor_p \right\rfloor_u$
return y_i

$\underline{\mathsf{Combine}(\{y_i\}_{i \in \mathcal{S}})}$

Run $\lambda_{i \in \mathcal{S}} = \mathsf{SS.CoefF}(\mathcal{S})$
Compute $Y' = \left\lfloor \sum_{i \in \mathcal{S}} \Delta\lambda_i \cdot y_i \right\rfloor_v$
return Y'

Figure 10: Construction of Distributed PRF based on the Secret Sharing scheme of Construction 6 where $\mathsf{SS} = (\mathsf{SecretShare}, \mathsf{Coeff})$ with $\mathsf{pp}_{\mathsf{SS}}$ denoting the public parameters of the secret sharing scheme.

[6] Then, the combine algorithm will simply multiply all partial evaluations with Δ as well.

Construction 13 (Distributed Almost Key Homomorphic PRF from $\mathtt{LWR}$)**.** A (t, m)-Distributed PRF is a tuple of PPT algorithms $\mathsf{DPRF} := (\mathsf{Gen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{P\text{-}Eval}, \mathsf{Combine})$ with the algorithms as defined in Figure 10.

Issues with the Construction from Boneh *et al.* [19, §7.1.1]. As remarked earlier, their generic construction suffers from issues stemming from their security reduction. Specifically, their security reduction proceeds similarly to the proof of Theorem 10 and requires $\mathcal{B}$ to answer honest evaluation queries for key indices for which it does not know the actual key share. Their explanation suggests that we again use the "clearing out the denominator" trick by multiplying with Δ. However, the issue is that the resulting response will be of the form $\Delta \cdot F(k_{i*}, x)$ for i^*, unknown to $\mathcal{B}$. Consequently, one has to change the partial evaluation response to also include this offset to ensure the correctness of reduction. This would imply that the $\mathsf{Combine}$ algorithm will multiply with Δ again, which would thus result in the actual Eval algorithm having an offset of Δ^2. Furthermore, the partial evaluation algorithm should also have to round down to the elements in $[0, u-1]$ for the same reason that the $\mathsf{Combine}$ algorithm required this fix.

Correctness.

$$\begin{aligned}
\left\lfloor \Delta \lfloor \langle \mathtt{H}(x), \mathbf{k} \rangle \rfloor_p \right\rfloor_u &= \left\lfloor \Delta \left\lfloor \sum_{z=1}^{\rho} H(x, z) \cdot \mathsf{k}_z \right\rfloor_p \right\rfloor_u \\
&= \left\lfloor \Delta \left\lfloor \sum_{z=1}^{\rho} H(x, z) \sum_{i_j \in \{i_1, \ldots, i_\mathsf{t}\}} \lambda_{i_j} s_z^{(i_j)} \right\rfloor_p \right\rfloor_u \\
&= \left\lfloor \Delta \left\lfloor \sum_{i_j \in \{i_1, \ldots, i_\mathsf{t}\}} \sum_{z=1}^{\rho} H(x, z) \cdot \lambda_{i_j} \cdot s_z^{(i_j)} \right\rfloor_p \right\rfloor_u \\
&= \left\lfloor \Delta \left\lfloor \sum_{i_j \in \{i_1, \ldots, i_\mathsf{t}\}} \langle \mathtt{H}(x), \lambda_{i_j} \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u
\end{aligned}$$

[6]However, their generic construction is incorrect, owing to issues in their security proof which is not entirely sketched out. We fix the issues in our construction.

$$= \left\lfloor \Delta \left(e_{\mathsf{t}} + \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \left\lfloor \lambda_{i_j} \langle \mathtt{H}(x), \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right) \right\rfloor_u$$

$$= \left\lfloor \Delta \left(e_{\mathsf{t}} + \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} e_{\lambda_{i_j}} + \lambda_{i_j} \left\lfloor \langle \mathtt{H}(x), \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right) \right\rfloor_u$$

$$= \left\lfloor \Delta \left(e_{\mathsf{t}} + \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} e_{\lambda_{i_j}} \right) + \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \Delta \cdot \lambda_{i_j} \left\lfloor \langle \mathtt{H}(x), \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u$$

$$= \left\lfloor \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \Delta \cdot \lambda_{i_j} \left\lfloor \langle \mathtt{H}(x), \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u$$

The last step follows provided the error term is small. Recall that $e_{\mathsf{t}} \in \{0, \ldots, \mathsf{t}\}$ and $e_{\lambda_{i_j}} \in \left\{0, \ldots, \lambda_{i_j}\right\}$. Now observe that we multiply with Δ and λ_{i_j} has a maximum value Δ. Therefore, $\Delta \cdot e_{\lambda i_j} < \Delta^2$. Therefore, the size of the error term is $\leqslant \mathsf{t} \cdot \Delta + \mathsf{t} \cdot \Delta^2$. Therefore, provided u is chosen such that $\lfloor p/u \rfloor > (\Delta + 1) \cdot \mathsf{t} \cdot \Delta$, then the last step is correct. Now, we have:

$$\left\lfloor \Delta \left\lfloor \langle \mathtt{H}(x), \mathbf{k} \rangle \right\rfloor_p \right\rfloor_u = \left\lfloor \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \Delta \cdot \lambda_{i_j} \left\lfloor \langle \mathtt{H}(x), \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u$$

Therefore,

$$\left\lfloor \Delta \left\lfloor \Delta \left\lfloor \langle \mathtt{H}(x), \mathbf{k} \rangle \right\rfloor_p \right\rfloor_u \right\rfloor_v = \left\lfloor \Delta \left\lfloor \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \Delta \cdot \lambda_{i_j} \left\lfloor \langle \mathtt{H}(x), \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u \right\rfloor_v$$

$$= \left\lfloor \Delta \left(e_{\mathsf{t}} + \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \lambda_{i_j} \left\lfloor \Delta \cdot \left\lfloor \langle \mathtt{H}(x), \mathbf{k}^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u \right) \right\rfloor_v$$

$$= \left\lfloor \Delta \left(e_{\mathsf{t}} + \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \lambda_{i_j} \mathsf{P\text{-}Eval}(\mathbf{k}^{i_j}, x) \right) \right\rfloor_v$$

$$= \left\lfloor \sum_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \lambda_{i_j} \cdot \Delta \cdot \mathsf{P\text{-}Eval}(\mathbf{k}^{(i_j)}, x) \right\rfloor_v$$

$$= \mathsf{Combine}(\left\{ \mathsf{P\text{-}Eval}(\mathbf{k}^{(i_j)}, x)_{i_j \in \{i_1,\ldots,i_{\mathsf{t}}\}} \right\})$$

provided $\lfloor u/v \rfloor > \mathsf{t}\Delta$.

Pseudorandomness. The proof of pseudorandomness follows the outline of the proof of Theorem 10 but with some important differences. First, we do not rely on integer secret sharing but rather plain secret sharing over the field. Therefore, the Lagrange coefficients correspond to λ_{i_j}. Or more formally, to respond to a partial evaluation query at point x_j with target key index $i*$, the adversary $\mathcal{B}$ does the following:

- Use its oracle to get partial evaluation on x_j at i_{t}, which we call as $h_{j,\mathsf{t}}$.
- Then, use Lagrange coefficients but with suitably multiplying with Δ to compute the correct distribution by rounding down to u. The choice of u guarantees that the response is correct.

For challenge query, it simply does two rounding down, first to u and then to v.

Verification of Almost Key Homomorphism. Let $\mathbf{k}_1, \mathbf{k}_2$ be two keys that are shared. Now, let the key shares received by some party i_j be $\mathbf{k}_1^{(i_j)}$ and $\mathbf{k}_2^{(i_j)}$. Then,

$$\mathsf{P\text{-}Eval}(\mathbf{k}_1^{(i_j)}, x) + \mathsf{P\text{-}Eval}(\mathbf{k}_2^{(i_j)}, x) = \left\lfloor \Delta \left\lfloor \langle H(x), \mathbf{k}_1^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u + \left\lfloor \Delta \left\lfloor \langle H(x), \mathbf{k}_2^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u$$

$$= \left\lfloor \Delta \left\lfloor \langle H(x), \mathbf{k}_1^{(i_j)} \rangle \right\rfloor_p + \Delta \left\lfloor \langle H(x), \mathbf{k}_2^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u - e_1$$
$$= \left\lfloor \Delta \left\lfloor \langle H(x), \mathbf{k}_1^{(i_j)} \rangle + \langle H(x), \mathbf{k}_2^{(i_j)} \rangle \right\rfloor_p \right\rfloor_u - 2e_1$$
$$= \mathsf{P\text{-}Eval}(\mathbf{k}_1^{(i_j)} + \mathbf{k}_2^{(i_j)}, x) - 2e_1$$

It follows that for n such keys:

$$\sum_{i=1}^{n} \mathsf{P\text{-}Eval}(\mathbf{k}_i^{(i_j)}, x) = \mathsf{P\text{-}Eval}(\sum_{i=1}^{n} \mathbf{k}_i^{(i_j)}, x) - n \cdot e_1$$

This shows that the P-Eval is almost key-homomorphic. Consequently, one can verify that the whole Eval procedure is almost key homomorphic for the appropriate error function. To do this, recall that from correctness of our algorithm:

$$\mathsf{Share}(\mathbf{k}, \mathsf{m}, \mathsf{t}) = \left\{\mathbf{k}^{(i)}\right\}_{i \in [\mathsf{m}]}, \mathsf{Combine}(\left\{\mathsf{Eval}(\mathbf{k}^{(i_j)}, x)\right\}_{i \in \{i_1, \ldots, i_t\}} = \mathsf{Eval}(\mathbf{k}, x)$$

In other words, for $\mathbf{k}_1, \mathbf{k}_2 \in \mathcal{K}$, $\mathsf{Share}(\mathbf{k}_1, \mathsf{m}, \mathsf{t}) = \left\{\mathbf{k}_1^{(i)}\right\}_{i \in [\mathsf{m}]}$ and $\mathsf{Share}(\mathbf{k}_2, \mathsf{m}, \mathsf{t}) = \left\{\mathbf{k}_2^{(i)}\right\}_{i \in [\mathsf{m}]}$, we will have for $i \in [\mathsf{m}]$, $\mathsf{Eval}(\mathbf{k}_1^{(i)}, x), \mathsf{Eval}(\mathbf{k}_2^{(i)}, x) = \mathsf{Eval}(\mathbf{k}_1^{(i)} + \mathbf{k}_2^{(i)}, x) - 2 \cdot e_1$.

D.2 One-shot Private Aggregation Construction based on LWR Assumption

We build OPA based on the LWR Assumption, building it based on the Key Homomorphic, Distributed PRF as presented in Construction 14. However, our construction is largely different from the template followed to build OPA from the $\mathtt{HSM}_\mathsf{M}$ assumption. This is primarily because of the growth in error when combining partial evaluations. Specifically, will get that $\mathsf{P\text{-}Eval}(\sum_{i=1}^{n} \mathbf{k}_i^{(j)}, x) = \sum_{i=1}^{n} \mathsf{P\text{-}Eval}(\mathbf{k}_i^{(j}, x) + e$ where $e \in \{0, \ldots, n-1\}$ where n is the number of clients participating for that label. This would require us to round down to a new value u' such that $\lfloor u/u' \rfloor > n-1$. Therefore, while we still employ the underlying functions of the distributed, key-homomorphic PRF based on LWR, we have to open up the generic reduction. For simplicity, we detail the construction for $L = 1$. Then, these are the differences:

- As done fore Construction 3, the input is encoded as $x_i \cdot n + 1$.
- Specifically, the client's share to the committee will only be the first level of the evaluation, i.e., rounded down to p.
- Then, committee member j will then add the shares up, multiply with the offset, and then round down to u. We will show that provided $\lfloor p/u \rfloor > \Delta \cdot n$, this is consistent with $\mathsf{P\text{-}Eval}(\sum_{i=1}^{n} \mathbf{k}_i^{(j)}, x)$.
- The decoding algorithm is also similar to the one from Construction 3.

Recall that DPRF correctness requires that $\lfloor p/u \rfloor > \mathsf{t} \cdot \Delta + \mathsf{t} \cdot \Delta^2$, and so one just needs $\lfloor p/u \rfloor > \max(\mathsf{t} \cdot \Delta + \mathsf{t} \cdot \Delta^2, n \cdot \Delta)$. Then, one can rely on the correctness of DPRF as shown below to argue that when the server runs DPRF.Combine, the output is $\mathsf{Eval}(\sum_{i=1}^{n} \mathbf{k}_i, x)$.

Correctness. We showed how the output of the server's invocation of DPRF.Combine is $\mathsf{Eval}(\sum_{i=1}^{n} \mathbf{k}_i, x)$. Now, let us look at the remaining steps:

$$X_\ell = \sum_{i=1}^{n} \mathsf{ct}_{i,\ell} - \mathsf{AUX}_\ell = \sum_{i=1}^{n} (x_{i,\ell} * n + 1) + \mathsf{Eval}(\mathbf{k}_i, \ell) - \mathsf{Eval}(\sum_{i=1}^{n} \mathbf{k}_i, \ell) \bmod v$$
$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n + \sum_{i=1}^{n} \mathsf{Eval}(\mathbf{k}_i, \ell) - \mathsf{Eval}(\sum_{i=1}^{n} \mathbf{k}_i, \ell) \bmod v$$
$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n + \mathsf{Eval}(\sum_{i=1}^{n} \mathbf{k}_i, \ell) - \mathsf{Eval}(\sum_{i=1}^{n} \mathbf{k}_i, \ell) - e_{n-1} \bmod v$$

$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e_{n-1} \bmod v$$

$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e_{n-1}$$

For the last step to hold, we need that

$$0 \leqslant n \cdot \sum_{i=1}^{n} x_{i,\ell} + n - e_{n-1} < v$$

e_{n-1} the small value is 0 and the largest value is $n-1$ which requires that $\sum_{i=1}^{n} x_{i,\ell} < (v-n)/n$. Note that we already require $\lfloor p/u \rfloor > n\Delta, \lfloor u/v \rfloor > \mathsf{t}\Delta \implies \lfloor p/v \rfloor > n\mathsf{t}\Delta^2$. In other words, $\sum x_i < \frac{p}{n^2 \mathsf{t} \Delta^2}$. Finally, $X'_\ell = n \cdot \sum_{i=1}^{n} x_{i,\ell} + n$ and that completes the remaining steps.

E Constructions of Leakage Resilient Key-Homomorphic Pseudorandom Functions

E.1 Construction from LWR Assumption

We also have the construction from Boneh *et al.* [19] of an *almost* Key Homomorphic PRF from LWR in the Random Oracle model which was later formally proved secure by Ernst and Koch [47] with $\gamma = 1$.

Construction 14 (Key Homomorphic PRF from LWR)**.** Let $\mathtt{H} : \mathcal{X} \to \mathbb{Z}_q^\rho$. Then, define the efficiently computable function $F_{\mathtt{LWR}} : \mathbb{Z}_q^\rho \times \mathcal{X} \to \mathbb{Z}_p$ as $\lfloor \langle \mathtt{H}(x), \mathbf{k} \rangle \rfloor_p$. $F_{\mathtt{LWR}}$ is an almost key homomorphic PRF with $\gamma = 1$.

Construction 15 (Length Extended Key-Homomorphic from LWR)**.** Let F_{LWR} be the function as defined in Construction 14. Then, consider $F^L_{\mathtt{LWR}} := (F_{\mathtt{LWR}}(\mathsf{k}, (x,1)), \ldots F_{\mathtt{LWR}}(\mathsf{k}, (x, L)))$.

It is easy to see that Construction 15 is a secure pseudorandom function. An adversary that can break the security of Construction 15 can be used to break the security of Construction 14.

Theorem 11 (Leakage Resilience of Construction 15)**.** *Let* PRF *be the PRF defined in Construction 15. Recall that* $\mathsf{PRF}.\mathcal{K} = \mathbb{Z}_q^\rho$. *Then, it is leakage resilient in the following sense:*

$$\{\mathsf{PRF}(\mathsf{k}, x), (\mathsf{k} + r) \bmod q : \mathsf{k}, r \leftarrow_\$ \mathcal{K}\} \approx_c \{Y, (\mathsf{k}+r) \bmod q : Y \leftarrow_\$ \mathcal{Y}, \mathsf{k}, r \leftarrow_\$ \mathcal{K}\}$$

Proof. The proof proceeds through a sequence of hybrids.

$\mathsf{Hybrid}_0(\kappa)$**:** The left distribution is provided to the adversary. In other words, the adversary gets:

$$\{\mathsf{PRF}(\mathsf{k}, x), (\mathsf{k} + r) \bmod q : \mathsf{k}, r \leftarrow_\$ \mathcal{K}\}$$

$\mathsf{Hybrid}_1(\kappa)$**:** In this hybrid, we replace $(\mathsf{k}+r) \bmod q$ with a uniformly random value $\mathsf{k}' \leftarrow_\$ \mathcal{K}$.

$$\left\{\mathsf{PRF}(\mathsf{k}, x), \mathsf{k}' : \mathsf{k}, \mathsf{k}' \leftarrow_\$ \mathcal{K}\right\}$$

Note that $(\mathsf{k}+r) \bmod q$ and k' are identically distributed. Therefore, the $\mathsf{Hybrid}_0, \mathsf{Hybrid}_1$ are identically distributed.

$\mathsf{Hybrid}_2(\kappa)$**:** In this hybrid, we will replace the PRF computation with a random value from the range.

$$\left\{Y, \mathsf{k}' : Y \leftarrow_\$ \mathcal{Y}, \mathsf{k}' \leftarrow_\$ \mathcal{K}\right\}$$

Under the security of the PRF, we get that $\mathsf{Hybrid}_1, \mathsf{Hybrid}_2$ are computationally indistinguishable.

$\mathsf{Hybrid}_3(\kappa)$**:** We replace k' with $(\mathsf{k}+r) \bmod q$.

$$\{Y, (\mathsf{k}+r) \bmod q : Y \leftarrow_\$ \mathcal{Y}, \mathsf{k}, r \leftarrow_\$ \mathcal{K}\}$$

As argued before $\mathsf{Hybrid}_2, \mathsf{Hybrid}_3$ are identically distributed.

Note that Hybrid_3 is the right distribution from the theorem statement. This completes the proof. □

E.2 Learning with Errors Assumption

Construction 16 (Key Homomorphic PRF from $\texttt{LWE}$)**.** Let $\texttt{H}_{\mathbf{A}} : \mathcal{X} \to \mathbb{Z}_q^{L\times\lambda}$. Then, define the efficiently computable function $F_{\texttt{LWE}} : (\mathbb{Z}_q^{\lambda} \times \chi^L) \times \mathcal{X} \to \mathbb{Z}_q^L$ as $F_{\texttt{LWE}}((\mathbf{k}, \mathbf{e}), x) := \texttt{H}_{\mathbf{A}}(x)\mathbf{k} + \mathbf{e}$. $F_{\texttt{LWE}}$ is an almost key homomorphic PRF.

Remark 8. We note that it has been shown that one can also sample $\mathbf{x}$ from the same distribution as $\mathbf{e}$. This is known as the short-secret LWE assumption and was employed in the encryption scheme of Lyubashevsky *et al.* [67]. An immediate consequence of this assumption is that one can set $\mathbf{A}$'s dimension m to be much smaller than what is required for the LWE assumption.

Similar to the $\texttt{LWR}$ construction, we need to show that the PRF based on $\texttt{LWE}$ assumption is also leakage resilient. Unfortunately, a similar proof technique does not work because the construction also suffers from leakage on the error vector which are usually Gaussian secrets. Instead, we rely on the following assumption:

F Deferred Proofs

Theorem 2. *Let κ_s and κ_c be the statistical and computational security parameters. Let L be the input dimension and n be the number of clients, that are* $\text{poly}(\kappa_c)$*. Let δ be the dropout threshold and η be the corruption threshold such that $\delta + \eta < 1$. Then, there exists an efficient simulator* Sim *such that for all* $\mathsf{K} \subset [n]$ *such that* $|\mathsf{K}| \leqslant \eta n$*, inputs* $X = \{\mathbf{x}_{i,\ell}\}_{i\in n\setminus\mathsf{K}}$*, and for all adversaries $\mathcal{A}$ against Construction 3, that controls the server and the set of corrupted clients* K*, which behave semi-honestly (resp. maliciously), the output of* Sim *is computationally indistinguishable from the joint view of the server and the corrupted clients.* Sim *is allowed to query* $F_{D,\delta}^{\ell}(X)$ *(defined in Equation 1) once, per iteration.*

Proof. We will prove the theorem statement by defining a simulator Sim, through a sequence of hybrids such that the view of the adversary $\mathcal{A}$ between any two subsequent hybrids are computationally indistinguishable. Let $H = [n]\setminus\mathsf{K}$, which are the set of honest clients. Further, let $\mathcal{C} = [n]\setminus D$ where D is the set of dropout clients.

It is important to note that the server is semi-honest. Therefore, it is expected to compute the set intersection of online clients $\mathcal{C}$, as expected. In other words, all committee members (and specifically the honest committee members) receive the same $\mathcal{C}$. This is an important contrast from active adversaries as a corrupt and active server could deviate from expected behavior and send different $\mathcal{C}^{(j)}$, for different committee members. This could help it glean some information about the honest clients.

We now sketch the proof below:

$\mathsf{Hybrid}_0(\kappa)$**:** This is the real execution of the protocol where the adversary $\mathcal{A}$ is interacting with the honest parties.

$\mathsf{Hybrid}_1(\kappa)$**:** In this hybrid, we will rely on the security of the secret sharing scheme to do two things:

- On the one-hand, all corrupt committee members receive a random share from the honest client's seed. Note that there can be only a maximum of t corrupt committee members. By appropriately choosing m, conditioned on η, we can guarantee that this holds with overwhelming probability. Then, for an honest client i, these are the shares denoted by $\left\{\mathsf{sd}_i^{(j)}\right\}_{j\in[\mathsf{m}]\cap\mathsf{K}}$ and are generated randomly.
- On the other hand, all the honest committee members receive a valid share of the honest client's seeds. However, each honest client i need to generate this from a polynomial $p(X)$ that satisfies $p(0) = \mathsf{sd}_i$, while also ensuring $p(j) = \mathsf{sd}_i^{(j)}$ for $j \in [\mathsf{m}] \cap \mathsf{K}$. Note that this is a polynomial time operation and is similar to the way packed secret sharing is done where multiple secrets are embedded at distinct points of the polynomial. See Construction 7 for how to build such a polynomial.

It is clear that by relying on the privacy of the secret sharing scheme, $\mathsf{Hybrid}_0, \mathsf{Hybrid}_1$ are indistinguishable for the adversary.

$\mathsf{Hybrid}_2(\kappa)$**:** In this hybrid, we change the definition of the last honest party's ciphertext. WLOG, let n be the last honest party in $\mathcal{C}$. Then, we will set $\mathbf{ct}_n := \mathsf{PRG.Expand}(\mathsf{sd}_\ell) + \mathbf{x}_\tau - \sum_{i \in \mathcal{C} \cap H} \mathbf{ct}_i$. Here, $\mathbf{x}_\tau$ is the sum of the honest clients inputs. Note that we are still in the hybrid where Sim knows all the inputs.

It is clear that $\mathsf{Hybrid}_1, \mathsf{Hybrid}_2$ are identically distributed, by the almost seed-homomorphism property of PRG, provided Sim chooses the inputs for the honest parties such that they sum up to the value in $\mathbf{x}_\tau$.

$\mathsf{Hybrid}_3(\kappa)$**:** Again, without loss of generality, let client 1 be the first honest client in $\mathcal{C} \cap H$. We will modify the way $\mathbf{ct}_{1,\ell}$ is generated. We will set it as $\mathbf{ct}_{1,\ell} := \mathbf{x}_{1,\ell} + u$ where $u \leftarrow\$ \mathsf{PRG}.\mathcal{Y}$.

$\mathsf{Hybrid}_2, \mathsf{Hybrid}_3$ are indistinguishable, provided Theorem 1 holds. In the reduction, we will implicitly set $\mathsf{sd}_1 + \mathsf{sd}_n$ to be the leakage obtained from the Theorem 1's challenger. In this hybrid, Sim still continues to know all the inputs. If it was a real PRG output, then we can simulate Hybrid_2, while simulating Hybrid_3 in the random case.

$\mathsf{Hybrid}_4(\kappa)$**:** In this hybrid, we will replace $\mathbf{ct}_{1,\tau} := u'$ for $u' \leftarrow\$ \mathsf{PRG}.\mathcal{Y}$. It is clear that $\mathsf{Hybrid}_3, \mathsf{Hybrid}_4$ are identically distributed.

At this point, observe that we have successfully replace the first honest client's ciphertext, with a uniformly random value that is independent of its input. Sim will continue to do this modification for every non-dropout honest client $i \in \mathcal{C} \cap H$. This leaves the clients with all-but-the-last honest clients' ciphertext to be independent of the input, while leaving the last honest client's ciphertext to be only a function of the sum of the inputs, which can be obtained by Sim's query to the functionality. Sim beings its interaction with the functionality. After all the honest clients have provided inputs to the trusted party $\mathcal{T}$, in Step (b), Sim does not instruct any corrupted client to abort but rather set their inputs to be 0. Then in Step (c), Sim does not abort the server. Therefore, in Step (d), Sim will learn the sum of the honest parties inputs. Denote it as $\mathbf{x}_\ell$, which is also the sum of the inputs of the honest, surviving clients. With this information, Sim uses the last hybrid to interact with the adversary $\mathcal{A}$, who's expecting the real world interaction. This will enable Sim to run $\mathcal{A}$ internally. This is crucial to ensure that Sim can get the output of $\mathcal{A}$, in the real world, which might depend on its view (including the output) of the server. This view will, in turn, depend on the the honest clients' inputs. Since Sim sets the honest inputs, in this internal execution, to match the sum of inputs in the real world, we can guarantee that the output of $\mathcal{A}$ in the internal simulation is indistinguishable from $\mathcal{A}$'s interaction in the real world by the aforementioned hybrid arguments.

Proof of Security against Active Server. Our constructions so far have relied on providing security against a semi-honest server. Note that, as shown in the proof of security for Theorems 2, we can use the functionality query to obtain the sum of all the honest non-drop out clients, as before.

In the semi-honest setting, it is easy to see that the set $\mathcal{C}$, with respect to which aggregation is performed, includes *all* the honest, non-dropout clients' inputs. Therefore, querying the functionality, Sim does indeed get the sum of all the honest clients' inputs that are also included in the summation in the real world. This is imperative to ensure that Sim, when internally invoking $\mathcal{A}$, can get the output of $\mathcal{A}$ which should be indistinguishable from $\mathcal{A}$'s output in the real world. Specifically, this output of $\mathcal{A}$ (in either the internal invocation or the actual execution) will depend on the view which consists of the output of the server. Therefore, if the output of the server in the real world does not include any of these honest clients' inputs, then the output produced by the internal invocation of $\mathcal{A}$ can be different from that in the real world.

Let us look at the case when the server is corrupted. Such a server can mount an attack whereby the real-world execution of the protocol may exclude inputs of some of those honest parties but actually included in the output of the ideal functionality. The proof of malicious security is tricky in this setting. Specifically, a malicious server can drop clients after seeing the honest input. This is an issue in the simulation as the simulator has to generate the masked inputs for the honest clients without knowing which of them would be dropped later.

Prior works, beginning with that of Bonawitz *et al.* [17] have relied on using signatures to ensure that a malicious server does not compromise the privacy of an honest user. Fortunately, for OPA, we can rely on the one-shot nature of communication flow to secure messages and avoid using signatures.

As before, let K denote the corrupted clients. Then, $\mathsf{H}_{\mathsf{Cli}} := [n]\backslash\mathsf{K}$ is the set of honest clients, $\mathsf{H}_{\mathsf{Com}} := [\mathsf{m}]\backslash\mathsf{K}$ is the set of honest committee members. Let $\mathsf{K}_{\mathsf{Cli}} := [n] \cap \mathsf{K}$ denote the corrupted clients and $\mathsf{K}_{\mathsf{Com}} := [\mathsf{m}] \cap \mathsf{K}$ denotes the corrupted committee members.

Note that we do not rely on signatures. To achieve a protocol with signatures, there needs to be an additional round of communication between the committee members and the server. First, the server forwards the message to the committee members. Then, the committee members responds with their set $\mathcal{C}^{(j)}$, which is also duly signed. Then, the server performs the intersection and contacts the committee member with this intersection along with signatures. A committee member then only aggregates if there are $(\mathsf{m}+\mathsf{t})/2$ valid signatures.

Our focus is to ensure that the committee members only speaks once. In other words, our construction currently has the server identify $\mathcal{C}^{(j)}$, for each $j \in [\mathsf{m}]$, based on the information it has received from the client. Then, the server forwards the message to the committee member along with its computed intersection. This setting allows the server to selectively forward shares to committee member and also choose different sets for different committee members. We will show that if $\mathsf{r} > (\mathsf{m}+\mathsf{t})/2$ where r is the reconstruction threshold in the committee and t is the corruption threshold, then the server doing so will receive meaningless information. Formally, we will show that there does not exist two sets of users $\mathcal{C} \neq \mathcal{C}'$ such that the server can reconstruct the shares over these two sets.

Observe that the server controls t committee members. We require each honest committee member to participate once, per iteration. This is easily enforced as the share from the honest client encrypts, along with the share for the honest committee member, also the identity of the honest client and the iteration count. Therefore, a server cannot replay the same share, in another iteration. With this guarantee, a malicious server, in order to reconstruct the shares of two distinct sets $\mathcal{C}, \mathcal{C}'$, will require the cooperation of at least $\mathsf{r}-\mathsf{t}$ honest users, while there are $\mathsf{r}-\mathsf{t}$ honest users present. We will therefore need $2(\mathsf{r}-\mathsf{t}) > \mathsf{m}-\mathsf{t}$. Or, $\mathsf{r} > (\mathsf{m}+\mathsf{t})/2$. This ensures that the server can only effectively reconstruct with respect to a unique set $\mathcal{C}$ and H^* is the set of honest users in this set. Note that the above inequality holds for $\mathsf{r} = 2 * \mathsf{m}/3, \mathsf{t} < \mathsf{m}/3$. Indeed, prior works such as Bonawitz *et al.* [17] and most recently LERNA [62] also tolerated only upto a $\mathsf{m}/3$ corruption threshold.

While we have shown that there is a unique set H^* of honest users, H^* is only revealed after all the honest clients have sent their inputs. Therefore, the simulator, during its internal execution of $\mathcal{A}$, needs to be able to generate the masked inputs for the honest users and it only knows the sum of *all* the honest clients that have not dropped out. This set may be distinct from H^*. Therefore, we need a way for the simulator to generate masked inputs, independent of the sum of the inputs, and then ensure that the correct sum is computed during reconstruction.

The simulator does the following:

- For every honest client that hasn't dropped out, i.e., for all $i \in \mathsf{H}_{\mathsf{Cli}}$, the simulator does the following:
  - Samples $\mathsf{dig}_{i,\ell} \leftarrow_\$ \{0,1\}^{\log q}$
  - Samples $\mathsf{sd}_{i,\ell} \leftarrow_\$ \mathsf{PRG}.\mathcal{K}$
  - It computes $\mathbf{mask}'_{i,\ell} := \mathtt{H}(\mathsf{dig}_{i,\ell}$
  - Computes $\mathbf{mask}_{i,\ell} = \mathsf{PRG.Expand}(\mathsf{sd}_{i,\ell})$
  - Sets $\mathbf{ct}_{i,\ell} := \mathbf{mask}_{i,\ell} + \mathbf{mask}'_{i,\ell}$
  - Like shown in proof of Theorem 2, the shares of $\mathsf{sd}^{(j)}_{i,\ell}, \mathsf{dig}^{(j)}_{i,\ell}$ for corrupt committee members $j \in \mathsf{K}_{\mathsf{Com}}$ are chosen at random. Meanwhile, the shares for the honest committee members are to be sampled in the second phase, with a specific purpose. However, the server still expects an encryption of shares from honest client to honest committee members. Therefore, it simply encrypts some random shares for the honest committee members too and sends it to the server.
  - It sends to $\mathcal{A}$, $\mathbf{ct}_{i,\ell}$ and $\mathsf{sd}^{(j)}_{i,\ell}$ and $\mathsf{dig}^{(j)}_{i,\ell}$ for $j \in [\mathsf{m}]$, which is encrypted appropriately.
- This concludes the client phase of the operation. Then, comes the interaction with the committee. Note that the simulator is also required to simulate the honest committee member j.
- The simulator, which has received $\mathcal{C}^{(j)}$ for each honest committee member j does the check to make sure that there exists at least $\mathsf{r}-\mathsf{t}$ such committee members with the same $\mathcal{C}^{(j)}$. We will call this client

set as $\mathcal{C}$, while calling the set of these committee members to be C_{good}. Meanwhile, it records those committee members with a different $\mathcal{C}(j)$. We will call this as some set C_{bad}. Looking ahead, for those honest committee members in C_{bad}, the shares of the honest clients that are to be added up is going to be random values. Note that $|C_{\mathsf{bad}}| \leqslant \mathsf{m} - \mathsf{r}$.

- The simulator now operates in two phases for honest committee member $j \in \mathsf{H}_{\mathsf{Com}}$. First is the share generation phase for honest clients i. It does the following:
  - If $j \in C_{\mathsf{bad}}$, then for honest client $i \in \mathcal{C}^{(j)} \cap \mathsf{H}_{\mathsf{Cli}}$, set $\mathsf{sd}_{i,\ell}^{(j)}, \mathsf{dig}_{i,\ell}^{(j)}$ to be random values.
  - Now, the simulator computes the shares for all honest clients i to $j \in C_{\mathsf{good}}$. These are valid shares of $\mathsf{sd}_{i,\ell}, \mathsf{dig}_{i,\ell}$ subject to the constraint that random values were fixed for those $j \in C_{\mathsf{bad}}$ where $i \in \mathcal{C}^{(j)}$, and for those $j \in \mathsf{K}_{\mathsf{Com}}$.
  - The honest committee member j receives from $\mathcal{A}$, $\mathsf{sd}_{i,\ell}^{(j)}$ and $\mathsf{dig}_{i,\ell}^{(j)}$ for $i \in \mathcal{C}^{(j)}$. Note that the maximum number of prefixed values is $\mathsf{m} - \mathsf{r} + \mathsf{t}$, and by our constraint $\mathsf{r} > \mathsf{m} - \mathsf{r} + \mathsf{t}$ which guarantees that these prefixed values cannot uniquely determine a polynomial of degree r.
- The second phase, is the combination phase. It responds, as expected, subject to the set $\mathcal{C}^{(j)}$ that it receives.
- Sim now queries the functionality. First, it provides $[n]\backslash\mathcal{C}$ as the set of dropped out clients. Then, it sends for those corrupted clients $\mathsf{K} \cap \mathcal{C}$, input as 0 to the functionality. In response, it gets $\sum_{\mathcal{C} \cap H} \mathsf{x}_{i,\ell}$. Call this x_H.
- Sim now picks $i^* \in \mathsf{H}^*$. It programs the random oracle by setting $\mathtt{H}(\mathsf{dig}_{i*,\ell}) = \mathsf{x}_H - \mathbf{mask}'_{i*,\ell}$.
- Sim continues to respond, on behalf of the honest committee member, as expected.
- Finally, $\mathcal{A}$ (which controls the server) will make queries to random oracle and it answers as expected. Sim outputs whatever $\mathcal{A}$ outputs at the end.

We will now need to show that the above simulation is indistinguishable from the real world execution that $\mathcal{A}$ expects when it is internally run. The hybrids proceeds as follows:

$\mathsf{Hybrid}_0(\kappa)$**:** This is the real world execution.

$\mathsf{Hybrid}_1(\kappa)$**:** In this execution, we replace the shares sent by the honest client i to honest committee member j, which are encrypted under pk_j with a random value. Under the semantic security of this encryption scheme, we can guarantee that this is indistinguishable from the previous hybrid. Meanwhile, these honest committee members (which the simulator controls) will receive the shares directly from the simulator. The view of $\mathcal{A}$, in this hybrid, is indistinguishable from the real world execution, under the semantic security of the encryption scheme.

$\mathsf{Hybrid}_2(\kappa)$**:** We will rely on the security of the secret sharing scheme to sample the shares for the honest clients, similar to Hybrid_1 of semi-honest security. For those $j \in \mathsf{K}_{\mathsf{Com}}$, the shares are randomly chosen. Furthermore, for those $j \in C_{\mathsf{bad}}$ also the shares are randomly chosen. Finally, for those $j \in C_{\mathsf{good}}$ it gets a valid share subject to those previously chosen random values. This is similar to Hybrid_1 in the proof of semi-honest security.

$\mathsf{Hybrid}_3(\kappa)$**:** In this hybrid, for all those honest clients i that are not in $\mathcal{C}$, we will set $\mathbf{ct}_{i,\ell} = \mathbf{mask}_{i,\ell} + \mathbf{mask}'_{i,\ell}$, effectively setting the input to be 0. Observe that the view of $\mathcal{A}$ remains unchanged as these honest clients inputs were never incorporated in the final sum anyway. Furthermore, if any of these $i \in \mathcal{C}^{(j)}$ for $j \in C_{\mathsf{bad}}$, the shares from these honest clients i to these j are completely random *and* independent of $\mathbf{mask}_{i,\ell}$ and $\mathbf{mask}'_{i,\ell}$.

$\mathsf{Hybrid}_4(\kappa)$**:** In this hybrid, we pick an honest surviving client $i^* \in \mathsf{H}^*$. It sets the inputs for all $i \neq i^* \in \mathsf{H}^*$ to be 0. Then sets $\mathsf{x}_{i*,\ell}$ to be the sum of all the $i \in \mathsf{H}^*$. Call this sum as x_H. Observe that the values are still correlated and pseudorandom.

$\mathsf{Hybrid}_5(\kappa)$: In this hybrid, we will program $\mathtt{H}(\mathsf{dig}_{i*,\ell}) = \mathbf{x}_H - \mathbf{mask}'_{i*,\ell}$, while setting $\mathbf{ct}_{i*,\ell} = \mathbf{mask}_{i*,lab} + \mathbf{mask}'_{i*,\ell}$. Note that because $\mathsf{dig}_{i*,\ell}$ is chosen uniformly at random from q values where q is a large prime. The probability of collision is negligible. There is only negligible difference in the view of $\mathcal{A}$.

$\mathsf{Hybrid}_6(\kappa)$: In this hybrid, we will set $\mathbf{ct}_{i,\ell}$ for $i \neq i^*$ to be some random term in the ciphertext space. Then, we will set $\mathbf{ct}_{i*,\ell} = \mathtt{H}(\mathsf{dig}_{i*,\ell}) - \sum_{i \neq i*} \mathbf{ct}_{i,\ell}$.

Note that under the leakage resilience property of the seed-homomorphic PRG, we can conclude that the two hybrids are computationally indistinguishable.

$\mathsf{Hybrid}_7(\kappa)$: In this hybrid, we will replace $\mathbf{ct}_{i,\ell} = \mathbf{mask}_{i,\ell} + \mathbf{mask}'_{i,\ell}$ for $i \neq i^*$.

Observe that this last hybrid is exactly what the simulator produces. This concludes the proof. □

F.1 Proof of Theorem 9

Theorem 9. *Construction 10 is a secure PRF where $\mathtt{H}$ is modeled as a random oracle under the $\mathtt{HSM}_{\mathsf{M}}$ assumption.*

Proof. We denote the challenger by $\mathcal{B}$. Let S_j be the event that the adversary wins in Hybrid_j for each $j \in \{0, \ldots, 2\}$. Let q_e (resp. q_h) denote the number of evaluation queries (resp. hash oracle queries) that the adversary makes. We use an analysis similar to the technique by Coron [38].

$\mathsf{Hybrid}_0(\kappa)$: Corresponds to the security game as defined for security of PRF. It follows that the advantage of the adversary is

$$\mathsf{Adv}_0 = 2 \cdot |\Pr[S_0] - 1/2] = \mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF}}$$

$\mathsf{Hybrid}_1(\kappa)$: This game is identical to Hybrid_1 with the following difference. The challenger tosses biased coin δ_t for each random oracle query $H(t)$. The biasing of the coin is as follows: takes a value 1 with probability $\frac{1}{q_e+1}$ and 0 with probability $\frac{q_e}{q_e+1}$. Then, one can consider the following event E: that the adversary makes a query to the random oracle with x_i as an input where x_i was one of the evaluation inputs and for this choice we have that δ_t was flipped to 0.

If E happens, the challenger halts and declares failure. Then, we have that:

$$\Pr[\neg E] = \left(\frac{q_e}{q_e+1}\right)^{q_e} \geqslant \frac{1}{e(q_e+1)}$$

where e is the Napier's constant. Finally, we get that:

$$\Pr[S_1] = \Pr[S_0] \cdot \Pr[\neg E] \geqslant \frac{\Pr[S_0]}{e(q_e+1)}$$

$\mathsf{Hybrid}_2(\kappa)$: This game is similar to Hybrid_1 with the following difference: we modify the random oracle outputs.

- If $\delta_t = 0$, the challenger samples $w_t \leftarrow\!\!\$\; \mathcal{D}_H$ and sets $H(t) = h^{w_t}$
- If $\delta_t = 1$, the challenger samples $w_t \leftarrow\!\!\$\; \mathcal{D}_H, u_t \leftarrow\!\!\$\; \mathbb{Z}/\mathsf{M}\mathbb{Z}$ and sets $H(t) = h^{w_t} \cdot f^{u_t}$

Note that, under the HSM assumption, an adversary cannot distinguish between the two hybrids. Therefore, we get:

$$|\Pr[S_2] - \Pr[S_1]| \leqslant \epsilon_{\mathtt{HSM}_{\mathsf{M}}}$$

where $\epsilon_{\mathtt{HSM}_{\mathsf{M}}}$ is the advantage that an adversary has in the $\mathtt{HSM}_{\mathsf{M}}$ game. Note that Hybrid_2 corresponds to the case where the outputs are all random elements in $\mathbb{G}$. Therefore, the inputs are sufficiently masked and leak no information about the key. Therefore, $\Pr[S_2] = 0$ Then,

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{PRF}} \leqslant (e \cdot (q_e+1) \cdot \epsilon_{\mathtt{HSM}_{\mathsf{M}}}$$

□

Remark 9. Note that the above scheme is simply an adaptation of the famous DDH-based construction of a key-homomorphic PRF that was shown to be secure by Naor *et al.* [71]. It is easy to verify that our construction is also key homomorphic as $H(x)^{(k_1+k_2)} = H(x)^{k_1} \cdot H(x)^{k_2}$.

F.2 Distributed Pseudorandom Function

Construction 11 (Distributed PRF in CL Framework)**.** A (r, m)-Distributed PRF is a tuple of PPT algorithms $\mathsf{DPRF} := (\mathsf{Gen}, \mathsf{Share}, \mathsf{Eval}, \mathsf{P\text{-}Eval}, \mathsf{Combine})$ with the algorithms as defined in Figure 8. For simplicity, in the construction below we will set the corruption threshold $\mathsf{t} = \mathsf{r} - 1$. Though, the construction also holds for a lower t.

Correctness. For a polynomial $f \in \mathbb{Z}[X]$, every $f(i)$ leaks information about the secret $\mathsf{s} \bmod i$ leading to a choice of polynomial f such that $f(0) = \Delta \cdot \mathsf{s}$. For our use case, the secret is the PRF key k. Let us consider a set $\mathcal{S} = \{i_1, \ldots, i_\mathsf{t}\}$ of indices and corresponding evaluations of the polynomial f at $i_1, \ldots, i_\mathsf{t}$ giving us key shares: $\mathsf{k}^{(i_1)}, \ldots, \mathsf{k}^{(i_\mathsf{t})}$. To begin with, one can compute the Lagrange coefficients corresponding to the set $\mathcal{S}$ as: $\forall\, i \in \mathcal{S}, \lambda_i(X) := \prod_{j \in \mathcal{S}\setminus\{i\}} \frac{x_j - X}{x_j - x_i}$. This implies that the resulting polynomial is $f(X) := \sum_{j=1}^{\mathsf{t}} \lambda_{i_j}(X) \cdot \mathsf{k}^{(i_j)}$.

However, $\lambda_i(X)$ requires one to perform a division $x_j - x_i$ which is undefined as $\mathtt{H}$ hashes to $\mathbb{G}$ whose order is unknown. To avoid this issue, a standard technique is to instead compute coefficient $\Lambda_i(X) := \Delta \cdot \lambda_i(x)$. Thereby, the resulting polynomial that is reconstructed if $f'(X) = \Delta \cdot f(X) = \sum_{j=1}^{\mathsf{t}} \Lambda_{i_j}(X) \cdot \mathsf{k}^{(i_j)}$. Consequently,

$$\begin{aligned}
\mathtt{H}(x)^{\Delta^3 \cdot \mathsf{k}} = \mathtt{H}(x)^{\Delta \cdot f'(0)} &= \mathtt{H}(x)^{\Delta \cdot \sum_{j=1}^{\mathsf{t}} \Lambda_{i_j}(0) \cdot \mathsf{k}^{(i_j)}} \\
&= \prod_{j=1}^{t} \left(\mathtt{H}(x)^{\Delta \cdot \mathsf{k}^{(i_j)}}\right)^{\Lambda_{i_j}(0)} \\
&= \prod_{j=1}^{t} \left(\mathsf{P\text{-}Eval}(\mathsf{k}^{(i_j)}, x)\right)^{\Lambda_{i_j}(0)}
\end{aligned}$$

Thus, our protocol is correct.

Pseudrandomness. Next, we consider the pseudorandomness property of our construction.

Theorem 10. *In the Random Oracle Model, if Construction 10 is a secure pseudorandom function if Integer Secret Sharing is statistically private, then Construction 11 is pseudorandom in the static corruptions setting.*

Boneh *et al.* [19] showed that from any Key Homomorphic PRF (which Construction 10), one can build a Distributed PRF. The proof of the following theorem follows the template of this scheme with certain important adaptations as our secret sharing scheme is over integers. The proof technique is to show that if there exists an adversary $\mathcal{A}$ that can break the DPRF security, one can then use it to build an adversary $\mathcal{B}$ to break the pseudorandomness of our original PRF, as defined in Construction 10. The idea behind the proof is for $\mathcal{B}$, upon receiving choice of $\mathsf{t} - 1$ corruptions as indices $i_1, \ldots, i_{\mathsf{t}-1}$, to then choose a random index i_t and implicitly set $\mathsf{k}^{(i_\mathsf{t})}$ to be the PRF key chosen by its challenger. Therefore, the $\mathcal{B}$ now has knowledge of t indices, with which it can sample the Lagrange coefficients as before:

for $j = 1, \ldots, \mathsf{t}$ **do**
    $\Lambda_{i_j}(X) := \prod_{\zeta \in \{1,\ldots,\mathsf{t}\}\setminus j}^{\mathsf{t}} \frac{i_\zeta - X}{i_\zeta - i_j} \cdot (\Delta)$

Now, $\mathcal{B}$ with knowledge of the keys for indices $i_1, \ldots, i_{\mathsf{t}-1}$ along with access to Oracle needs to simulate valid responses to $\mathsf{P\text{-}Eval}$ queries for an unknown index. Call this index i^*. Then, we have:

$$\begin{aligned}
\mathsf{P\text{-}Eval}(\mathsf{k}^{(i^*)}, x) : = \mathtt{H}(x)^{\Delta \cdot \mathsf{k}^{(i^*)}} &= \mathtt{H}(x)^{\sum_{j=1}^{\mathsf{t}} \Lambda_{i_j}(i^*) \cdot \mathsf{k}^{(i_j)}} \\
&= \mathtt{H}(x)^{\sum_{i=1}^{\mathsf{t}-1} \Lambda_{i_j}(i^*) \cdot \mathsf{k}^{(i_j)}} \cdot \left(\mathtt{H}(x)^{\mathsf{k}^{(i_\mathsf{t})}}\right)^{\Lambda_{i_\mathsf{t}}(i^*)}
\end{aligned}$$

The last term is simulated using $\mathcal{B}$'s own oracle access.

Proof. Let $\mathcal{A}$ be a PPT attacker against the pseudorandomness property of DPRF, having advantage ϵ.

$\mathcal{A}$ first chooses $\mathsf{t} - 1$ indices $\mathbf{K} = \{i_1, \ldots, i_{\mathsf{t}-1}\}$ where each index is a subset of $\{1, \ldots, \mathsf{m}\}$. $\mathcal{A}$ receives the shares of the keys $\mathsf{k}^{(i_1)} = f(i_1), \ldots, \mathsf{k}^{(i_{\mathsf{t}-1})} = f(i_{\mathsf{t}-1})$ (for unknown polynomial f of degree t such that $f(0) =$

$\mathsf{k} \cdot \Delta$ with $\Delta := \Delta$. Further, $\mathcal{A}$ has access to $\mathcal{O}_{\mathsf{Eval}}(i, x)$ receiving $\mathsf{P\text{-}Eval}(\mathsf{k}_i, x)$ ins response. Additionally, $\mathcal{A}$ expects to have oracle access to the random oracle $\mathtt{H}$.

Using this attacker $\mathcal{A}$, we now define a PPT attacker $\mathcal{B}$ which will break the pseudorandomness property of Construction 10. Note that $\mathcal{B}$ is given access to the oracle that either outputs the real evaluation of the PRF on key k^* or a random value. Additionally, $\mathcal{B}$ expects to have oracle access to the random oracle $\mathtt{H}$.

- **Setup:** $\mathcal{B}$ does the following during Setup.
  - Receive set $S = \{i_1, \ldots, i_{\mathsf{t}-1}\}$ from $\mathcal{A}$.
  - Next $\mathcal{B}$ generates the key shares and public key as follows:
    * Sample $\mathsf{k}^{(i_1)}, \ldots, \mathsf{k}^{(i_{\mathsf{t}-1})} \in \mathbb{Z}$.
    * $\mathcal{B}$ picks an index i_{t} at random and implicitly sets the PRF key chosen by its challenger as $\mathsf{k}^{(i_t)}$.
    * Immediately, given the t indices, one can construct the secret sharing polynomial $f \in \mathbb{Z}[X]$ as described earlier, but instead recreating the polynomial $f'(X)$ using the coefficients $\Lambda_{i_j}(X)$ for $j = 1, \ldots, \mathsf{t}$ with $\mathsf{k}^{(i_t)}$ being unknown to $\mathcal{B}$ and using its challenger to simulate a response.
    * $\mathcal{B}$ gives $\mathsf{k}^{(i_1)}, \ldots, \mathsf{k}^{(i_{\mathsf{t}-1})}$ to $\mathcal{A}$.
- **Queries to H:** $\mathcal{B}$ merely responds to all queries from $\mathcal{A}$ to $\mathtt{H}$ by using its oracle access to $\mathtt{H}$.
- **Queries to Partial Evaluation:** $\mathcal{B}$ receives as query input, some choice of key index specified by i^* and input x_j for $i = 1, \ldots, Q$. In response $\mathcal{B}$ does the following:
  - Forward x_j to its challenger. In response it implicitly receives $\mathsf{P\text{-}Eval}(\mathsf{k}^{(i_{\mathsf{t}})}, x_j)$, but off by a factor of Δ in the exponent. Call this $h_{j,\mathsf{t}}$.
  - Compute: $h_{j,i^*} = \mathtt{H}(x_j)^{\sum_{i=1}^{\mathsf{t}-1} \Lambda_{i_j}(i^*) \cdot \mathsf{k}^{(i_j)}} \cdot (h_{j,\mathsf{t}})^{\Lambda_{i_{\mathsf{t}}}(i^*)}$ where $\mathcal{B}$ uses its own access to hash oracle to get $\mathtt{H}(x_j)$.
  - It returns h_{j,i^*} to $\mathcal{A}$.
- **Challenge Query:** On receiving the challenge input x^*, $\mathcal{B}$ does the following:
  - Ensure that it is a valid input, i.e., there is no partial evaluation queries on x^* at any unknown index point.
  - If not, $\mathcal{B}$ forwards to its challenger x^*. In response it implicitly receives $\mathsf{P\text{-}Eval}(\mathsf{k}^{(i_{\mathsf{t}})}, x^*)$, but off by a factor of Δ in the exponent. Call this h^*.
  - It also uses its oracle access to $\mathtt{H}$ to receive $h = \mathtt{H}(x^*)$.
  - It finally computes $y = \mathtt{H}(x_j)^{\Delta^2 \cdot \sum_{i=1}^{\mathsf{t}-1} \Lambda_{i_j}(0) \cdot \mathsf{k}^{(i_j)}} \cdot (h^*)^{\Delta^2 \cdot \Lambda_{i_{\mathsf{t}}}(0)}$ and outputs y to $\mathcal{A}$
- **Finish:** It forwards $\mathcal{A}$'s guess as its own guess.

Analysis of the Reduction. Note that for the case when $b = 0$, $\mathcal{A}$ expects to receive $\mathtt{H}(x^*)^{\Delta^3 \cdot \mathsf{k}}$ where k is defined at the point 0. So, we get:

$$
\begin{aligned}
\mathtt{H}(x^*)^{\Delta^3 \cdot \mathsf{k}^{(0)}} &= \mathtt{H}(x^*)^{\Delta^2 \cdot f'(0)} = \mathtt{H}(x)^{\Delta^2 \sum_{j=1}^{\mathsf{t}} \Lambda_{i_j}(0) \cdot \mathsf{k}^{(i_j)}} \\
&= \mathtt{H}(x)^{\Delta^2 \sum_{i=1}^{\mathsf{t}-1} \Lambda_{i_j}(0) \cdot \mathsf{k}^{(i_j)}} \cdot \left(\mathtt{H}(x)^{\mathsf{k}^{(i_{\mathsf{t}})}}\right)^{\Delta^2 \Lambda_{i_{\mathsf{t}}}(0)}
\end{aligned}
$$

This shows that the returned value y is consistent when $b = 0$. Meanwhile, when $b = 1$, h^* is a random element in the group and then y is a truly random value which means that $\mathcal{B}$ has produced a valid random output for $\mathcal{A}$. Similarly, when $b = 0$, every response to partial evaluation is also done consistently by correctness of the underlying secret sharing scheme. Meanwhile, when $b = 1$, we can rely on the statistical privacy preserving guarantee of the underlying secret sharing scheme to argue that the difference that the adversary can notice is statistically negligible. This concludes the proof where $\mathcal{B}$ can only succeed with advantage ϵ. □

G Stronger Security Definition

Hitherto, we have only considered the indistinguishability of information from the perspective of the server. However, one can consider the requirement to hold for even corrupt committee members. Specifically, should their entire committee collude (or at least t of them), then the client's input remains hidden. It is easy to observe that Figure 9 does not satisfy the stronger security definition. Specifically, if we had a single committee member, then the auxiliary information (which is available to the committee member) simply masks the input and therefore can be unmasked. Thus, to accommodate security against collusion of all committee members we modify OPA syntax and construction to include a key from the server to keep client privacy. Informally, we do the following:

- First, the server or the aggregator has a secret key as well. This is denoted by k_0.
- Second, for each label ℓ, the server first publishes a "public key", as a function of the following algorithm $\mathsf{aux}_{0,\ell} \leftarrow \mathsf{PublicKeyGen}(\mathsf{k}_0, \ell)$
- Third, the encrypt procedure takes into account this auxiliary information, i.e., $\left(\mathsf{ct}_{i,\ell}, \left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in[\mathsf{m}]}\right)$ $\leftarrow_\$ \mathsf{Enc}(\mathsf{pp}, \mathsf{k}_i, x_i, \mathsf{aux}_{0,\ell}, \mathsf{t}, \mathsf{m}, \ell)$. In other words,the server publishes the public key and then the client can begin encrypting to a label.
- Fourth, the decrypt procedure takes k_0 as input too.

The committee indistinguishability game proceeds in phases.

- Setup Phase: The challenger begins by running the setup algorithm to generate the system parameters. The adversary is then provided with the system parameters pp and is asked to output an adversarial choice of n, which is the number of users that will be registered. In response, the challenger runs the KeyGen algorithm $n+1$ times, once each for the n users and once for the server's secret key. This phase ends with the adversary being provided with the server's secret key denoted by k_0.
- Learning Phase: The adversary issues queries to the various oracles defined by $\mathcal{O}_{\mathsf{Corr}}, \mathcal{O}_{\mathsf{Enc}}$ to learn any information it could. $\mathcal{O}_{\mathsf{Corr}}$ proceeds where the adversary can corrupt any user and receive its key. These corruptions are tracked. Meanwhile, $\mathcal{O}_{\mathsf{Enc}}$ allows the adversary to issue any arbitrary encryption queries on behalf of any of the users, with the restriction that it can only do so once per user per label. in response, it receives both the ciphertext encrypting the input and *all* the auxiliary information.

  This phase ends with the adversary committing to a target label τ.
- Challenge Phase: In this phase, the challenger begins by identifying eligible users $\mathcal{U}$ who are honest, which is defined by $[n]\backslash\mathsf{K}$. Without loss of generality, we assume that there have been no queries to $\mathcal{O}_{\mathsf{Enc}}$ with τ as the label. Should there be such queries, those users i such that $(i, \tau, \cdot) \in \mathbf{E}$ are also removed from the set $\mathcal{U}$ and these inputs are later used to compute the challenge. Upon receiving, $\mathcal{U}$, the adversary commits to two sets: $\mathcal{H} \subseteq \mathcal{U}$ is the set of honest users that the adversary is targeting, and $\mathcal{S}$ that is the set of committee members for whom the adversary receives $\left\{\mathsf{aux}_{i,\tau}^{(j)}\right\}_{i\in\mathcal{H}, j\in\mathcal{S}}$ provided $|\mathcal{S}| \leqslant \mathsf{t}-1$. Further, the adversary also provides inputs two choices of inputs for user in $\mathcal{H}$ denoted by $\{x_{i,0}, x_{i,1}\}_{i\in\mathcal{H}}$ and inputs $\{x_i\}_{i\in[n]\backslash\mathcal{H}}$ for the remaining users.
- Finally, the adversary is provided with individual encryptions and auxiliary information for all committee members.
- Guessing Phase: The adversary outputs a guess b' and wins if $b' = b$, provided trivial attacks do not happen.

Definition 12 (C-IND-CPA Security)**.** *We say that a* $(\mathsf{t}, \mathsf{m}, \mathsf{M})$ *One-shot Private Aggregation Scheme* OPA *with label space* $\mathcal{L}$ *is Server-Indistinguishable under Chosen Plaintext Attack (S-IND-CPA) if for any PPT* *adversary $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{pp} \leftarrow\!\!\$\; \mathsf{Setup}(1^\kappa); b \leftarrow\!\!\$\; \{0,1\} \\ (st, n) \leftarrow\!\!\$\; \mathcal{A}(\mathsf{pp}), \{\mathsf{k}_i \leftarrow\!\!\$\; \mathsf{KeyGen}()\}_{i\in[n]\cup\{0\}} \\ (st, \tau) \leftarrow\!\!\$\; \mathcal{A}^{\mathcal{O}_{\mathsf{Corr}},\mathcal{O}_{\mathsf{Enc}}}(st, \mathsf{k}_0), \mathcal{U} := [n]\backslash\mathsf{K} \\ (\mathcal{H}, \mathcal{S}, \{x_{i,0}, x_{i,1}\}_{i\in\mathcal{H}}, \{x_i\}_{i\in[n]\backslash\mathcal{H}}) \leftarrow\!\!\$\; \mathcal{A}(st, \mathcal{U}) \\ \left\{\mathsf{ct}_{i,\tau}, \left\{\mathsf{aux}_{i,\tau}^{(j)}\right\}_{j\in[\mathsf{m}]} \leftarrow\!\!\$\; \mathsf{Enc}(\mathsf{pp}, r_i, x_{i,b})\right\}_{i\in\mathcal{H}} \\ \left\{\mathsf{ct}_{i,\tau}, \left\{\mathsf{aux}_{i,\tau}^{(j)}\right\}_{j\in[\mathsf{m}]} \leftarrow\!\!\$\; \mathsf{Enc}(\mathsf{pp}, r_i, x_i)\right\}_{i\in[n]\backslash\mathcal{H}} \\ b' \leftarrow\!\!\$\; \mathcal{A}(st, \left\{\mathsf{ct}_{i,\tau}, \mathsf{aux}_{i,\tau}^{(j)}\right\}_{i\in[n], j\in[\mathsf{m}]}) \end{array} \right] \leqslant \frac{1}{2} + \mathrm{negl}(\kappa)$$

G.1 Updated Committee Indistinguishable Construction

These are the changes to OPA construction based on the $\mathtt{HSM}_\mathsf{M}$ assumption to adapt it to the stronger security definition:

- $\underline{\mathsf{PublicKeyGen}(\mathsf{k}_0, \ell)}$

  Compute $\mathsf{pk}_{0,\ell} \leftarrow \mathsf{DPRF.Eval}(\mathsf{k}_0, \ell)$
  return $\mathsf{pk}_{0,\ell}$

- Modify the encryption procedure as follows:

  $\underline{\mathsf{Enc}(\mathsf{pp}, \mathsf{k}_i, x_i, \mathsf{pk}_{0,\ell}, \mathsf{t}, \mathsf{m}, \ell)}$

  Parse $\mathsf{k}_i = \mathsf{k}_i$
  Compute $h_{i,\ell} = \mathsf{DPRF.Eval}(\mathsf{k}_i, \ell)$
  Compute $\mathsf{ct}_{i,\ell} = f^{x_i} \cdot \mathsf{pk}_{0,\ell}^{\mathsf{k}_i}$
  Compute $(\mathsf{k}_i^{(j)})_{j\in[\mathsf{m}]} \leftarrow\!\!\$\; \mathsf{DPRF.Share}(\mathsf{k}_i, \mathsf{t}, \mathsf{m})$
  for $j = 1, \ldots, \mathsf{m}$ **do**
  $\quad\mathsf{aux}_{i,\ell}^{(j)} = \mathsf{DPRF.Eval}(\mathsf{k}_i^{(j)}, \ell)$
  return $\mathsf{ct}_{i,\ell}, \left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in[\mathsf{m}]}$

  In other words, $\mathsf{aux}_{i,\ell}^{(j)}$ can be viewed as the j-th partial evaluation of the key (k_i) where k_i was key shared using Secret Sharing scheme, while $\mathsf{ct}_{i,\ell}$ was masked by the DPRF evaluation on the key $\mathsf{k}_i \cdot \mathsf{k}_0$

Now, observe that the C-Combine algorithm merely multiplies all the auxiliary information. As a result, $\mathsf{AUX}_\ell^{(j)}$ is simply a partial evaluation of the following key share $\sum_{i=1}^n \mathsf{k}_i^{(j)}$). Therefore, S-Combine computes $\mathsf{DPRF.Eval}(\cdot(\sum_{i=1}^n \mathsf{k}_i)$. Now, let us look at the decryption procedure:

$\underline{\mathsf{Aggregate}(\mathsf{AUX}_\ell, \mathsf{k}_0\, \{\mathsf{ct}_{i,\ell}\}_{i\in\mathcal{C}})}$

Compute $M = \mathsf{AUX}_\ell^{-\mathsf{k}_0} \cdot (\prod_{i\in\mathcal{C}} \mathsf{ct}_{i,\ell})$
Compute $X_\ell \leftarrow \mathsf{CLSolve}(\mathsf{pp}, M)$
return $X_\ell \bmod \mathsf{M}$

The security of this construction follows from the intuition that the adversary gets all of the auxiliary information, from which it can only construct a Diffie-Hellman key on the fly, from which it cannot compute any masking information to unmask the inputs.