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

May 11, 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 One-shot Private Aggregation (OPA) where clients speak only once (or even choose not to speak) per aggregation evaluation. Since every client communicates just once per aggregation, this streamlines the management of dropouts and dynamic participation of clients, contrasting with multi-round state-of-the-art protocols for each aggregation.

We initiate the study of OPA in several ways. First, we formalize the model and present a security definition. Second, we construct OPA protocols based on class groups, DCR, and LWR assumptions. Third, we demonstrate OPA with two applications: private stream aggregation and privacy-preserving federated learning. Specifically, OPA can be used as a key building block to enable privacy-preserving federated learning and critically, where client *speaks once*. This is a sharp departure from prior multi-round 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 leverage OPA to develop a streaming variant named SOPA, serving as the building block for privacy-preserving federated learning. We utilize SOPA to construct logistic regression classifiers for two datasets.

A new distributed key homomorphic PRF is at the core of our construction of OPA. This key component 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 distributed key homomorphic PRFs based on class groups or DCR or the LWR assumption.

Contents

1 Introduction

Minimizing interaction in Multiparty Computation (MPC) stands as a highly sought-after objective in the field of 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. This prompts the crucial question: Is it possible to further reduce interaction? Generally, the answer is negative. The underlying reason is that any non-interactive protocol, which operates with a single communication round, becomes susceptible to a vulnerability referred to as the "residual attack" [60] 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.

In this work, we explore 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 outputs of the aggregations they participated in. This holds for any polynomial number of computation sessions. See Section 2 for a comparison to other models.

2 Technical Survey of Related Work

Multi-Round Private Summation. We begin by revisiting the concept outlined in [60]. 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 [88], 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 [63, 13], and lattice-based cryptography [10, 53, 91, 90, 95].

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 [23] 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 [38, 50, 82] to build their distributed setup PSA. Meanwhile, the work of Nguyen *et al.* [78], 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 Multiclient 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 [68] 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). In Federated Learning, 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.

That said, 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. [18] and Bell et al. [12] 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 [64] 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 [12, 70, 57, 74, 11, 72, 73] of [18] 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 [42], facilitating data analytics for digital contact tracing, as seen in [5] besides enabling secure federated learning.

It is also important to note that some of these works - ACORN [11] and RoFL [73] build on top of the works of [18, 12] 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. In Section F, we show how to add to OPA to catch malicious clients, we leave it as a direction for future research on how to augment our protocol to also support input validation.

In Section 3, we present our contributions. The above discussion is also summarized in Table 1, by looking at four properties (a) whether the aggregate can be efficiently recovered, (b) whether it allows dynamic participation, (c) whether it requires trusted setup or multi-round distributed setup, and (d) the security assumptions.

2.1 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. [59]. 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 [55, 43, 3, 83, 99], have progressed to a point where their potential standardization by the IETF, as mentioned in [80], 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 [84] and SuperFL [99] or the generic multi-server solution Flag [9]. 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,

Table 1: 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, [88, 13, 57] require some restrictions on input sizes to recover the aggregate due to the discrete logarithm bottleneck.

	Efficient Aggregate	**Dynamic Participation**	TD vs DS	**Assumptions**
[88]	✗	✗	TD	DDH
[63]	✓	✗	TD	DCR
[13]	✗/✓	✗	TD	DDH/DCR
[10]	✓	✗	TD	LWE/R-LWE
[92]	✓	✗	TD	R-LWE
[95]	✓	✗	TD	AES
[90]	✓	✗	TD	RLWE
[68]	✗	✓	TD	DCR
[53]	✓	✗	TD	LWR
[23]	✗	✗	DS	LWR
[18, 12]	✓	✓	DS*	DDH
[57]	✓	✓	DS	DDH
[74]	✓	✓	DS	DDH
[70]	✓	✓	DS	DDH
Our Work	✓	✓	NA	HSM, LWR

where an adversary can corrupt parties after committee selection. The YOSO model, introduced by Gentry *et al.* [37] 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 [40, 17] 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.

3 Our Contributions

We initiate the study of OPA at the following fronts:

- **Modeling:** Formally, we introduce the OPA model of operation and formalize a *simple* game-based security notion. Our model is 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. In our modeling, we associate a committee with each iteration. The client communicates speaks once - sending a message to the committee and the server. The committee members communicate with the server to aid the summation. This diverges from prior approaches [18, 12, 70, 74, 57], which necessitate multiple interaction rounds and the management of dropout parties to handle communication delays or lost connections.
- **Cryptographic Assumptions:** We construct the first OPA protocols providing a suite of five distinct versions based on a diverse spectrum of assumptions:
  - Hidden Subgroup Membership ($\texttt{HSM}_{\mathsf{M}}$) asusmption where M is a prime integer.
  - $\texttt{HSM}_{\mathsf{M}}$ assumption where $\mathsf{M} = p^k$ for some prime p and integer k.
  - $\texttt{HSM}_{\mathsf{M}}$ assumption where $\mathsf{M} = 2^k$
  - $\texttt{HSM}_{\mathsf{M}}$ assumption where $\mathsf{M} = N$ where N is an RSA modulus (i.e., the DCR assumption)
  - Learning With Rounding (LWR) Assumption
- **Threat Models:** Our protocols do 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 present various levels of trust.
  - First, we allow the server to be corrupt, corrupt clients, and corrupt up to a certain threshold t of the committee members where t is the privacy threshold for secret reconstruction. Note that malformed encryptions by malicious client can be viewed as the client changing its input, for federated learning applications.
  - Second, we strengthen the security to allow for the compromise of all the committee members (while ensuring the server is not corrupt) and we present constructions that satisfy this stronger definition.
  - Third, we present a version of the protocol that is secure against active adversaries. Like previous work, this relies on signatures. This adds an additional round-trip communication between committee and server, while the client still speaks once. This specifically covers attacks where a malicious server tries to drop clients, based on their inputs.
  - Fourth, we present a robust version against malicious clients and/or committee members with the goal of identifying and removing these malicious clients' inputs from the aggregation. This relies on the client attaching a "proof" of honest behavior in communication to both server and committee. The client still speaks once.
- **Modes of Operations:** Our protocol allows to be operated in two modes - "new key" and "key reuse". In the latter, the clients reuse the keys, across various iterations. Meanwhile, we can leverage the non-interactive nature of our protocol to have the clients generate new keys, in every iteration, and the committee members receive a share of a new key, as opposed to some mapping of the key. The New Key mode offers several attractive features: (a) in some settings this has better communication as a field element can be sent, as opposed to a group element, (b) quick recovery from any key compromise as a new key is used in every round.

Meanwhile, prior works (especially LERNA) require a expensive setup phase to be performed when a new key needs to be established. In LERNA, this involves speaking with a committee of size 2^{14}.

On the other hand, the new key mode does reveal to the server the sum of the keys and can constitute a big leakage if clients are not careful about sampling new keys. Meanwhile, key reuse mode usually produces only an "ephemeral" encoding of the sum of the keys.

- **List of Techniques:**
  - We build OPA constructions based on new Distributed, Key Homomorphic PRFs. We operate both in the CL framework [34] and the LWR-based assumption.
  - We build the first Key Homomorphic PRF based on the $\mathtt{HSM}_{\mathsf{M}}$ assumption. We rely on Shamir Secret Sharing [87] over integers we also present Distributed, Key Homomorphic, PRFs.
  - We extend the almost Key Homomorphic PRF based on the LWR assumption [20, 53] 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.* [20]
  - 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 robust variant of OPA leveraging the Verifiable Secret Sharing over Integers [22].
  - Of independent interest, we also build a seed homomorphic PRG from $\mathtt{HSM}_{\mathsf{M}}$ assumption for the most efficient secure aggregation protocol we present in the context of federated learning.

- **Applications:** We showcase the power of OPA protocols in two practical applications
  - First, OPA empowers secure stream aggregation that was first introduced by Shi *et al.* [88] without necessitating the use of a Public Key Infrastructure (PKI) or the Common Reference String (CRS) model, prerequisites of prior techniques. Additionally, OPA supports dynamic participation. Independently, OPA finds utility in distributed multi-client functional encryption [78], eliminating the need for a setup phase, thereby enhancing its efficiency and practicality.
  - Second, leveraging the dynamic participation feature of OPA, crucial for federated learning, it 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 responds with *one* message corresponding to the new updates generated after the client has trained the model with its local data. In contrast, prior works[18, 12] involve 8 rounds, and the work of [74] requires 7 rounds in total, including the setup. Moreover, our protocols offer adaptive security. Our advantages extend beyond just round complexity. The server's asymptotic computation cost is $O(nL + L(\log n \log\log n))$ while that of Flamingo is $O(nL + n\log^2 n)$ and that of [12] is $O(n\log^2 n + nL\log n)$. The client computation cost is $O(L + \log n)$ (with committee members doing $O(n)$ work) while Flamingo has clients' computation at $O(L + n\log n)$. Similar asymptotic gains can be observed in the communication costs. Our federated learning protocols stand out by dramatically reducing the number of communication rounds, ensuring that each participant's engagement is just one round of interaction, thereby bridging the gap between learning in the clear and learning with secure aggregation. (See section 2 for a detailed comparison).

- **Implementation and Benchmarks:** Our contributions extend beyond the theoretical domain. We implement OPA as a secure aggregation protocol and benchmark with several state-of-the-art solutions [18, 12, 57, 74]. Specifically, we implement that OPA protocol that is 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 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. Note that the motivating application of the secure aggregation protocol remains federated learning. Therefore, to 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. More details can be found in Section 10.

3.1 Detailed Contributions in Federated Learning

Next, we compare our protocol with all efficient summation protocols listed in Table 1, with a specific focus on those that accommodate dynamic participation, a key feature shared by all federated learning methodologies.

We introduce SOPA where users/clients speak only once, significantly reducing the round complexity and overhead associated with each training iteration of prior works. SOPA is the streaming version of OPA, designed to handle a vector of inputs, per client. This innovative approach streamlines the federated learning process, making it more efficient and less resource-intensive, ultimately contributing to improved model training and accuracy. We propose three new secure aggregation protocols for federated learning:

- SOPA_1 where the committee size is set to be $\log n$. Note that our protocols can support any size of committees, but we set m to be $\log n$ for ease of presentation. This follows prior work such as [12] where they have each client secret-share their keys with a "neighborhood" of clients of size $\log n$,
- SOPA_1^p, which is SOPA_1 based on packed secret sharing for better performance
- SOPA_2, based on a seed-homomorphic PRG, offering better efficiency for large input vectors

In our threat model, we tolerate any corruption of users and we offer two flavors: one in which the server colludes with a maximum of $t - 1$ committee members (where t denotes the privacy threshold), and the other where all committee members are allowed to collude while maintaining privacy. The latter flavor is not offered by any prior multi-round protocol based on special groups of users such as [74, 11, 57, 70, 12], and does not change the communication or the computation cost.

In Tables 2 and 3, 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 2, 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

Table 2: Total asymptotic computation cost for all rounds per aggregation with semi-honest security. n denotes the total number of users, with committee size $\mathsf{m} = \log n$ 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 in 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.

Protocol	**Rounds**		**Computation Cost**			
			Server		User	
	Setup	Agg.	Setup	Agg.	Setup	Agg.
BIK+17[18]	-	8	-	$O(n^2L)$	-	$O(n^2 + nL)$
BBG+20[12]	-	8	-	$O(n \log^2 n + nL \log n)$	-	$O(\log^2 n + L \log n)$
Flamingo[74]	4	3	fwd	$O(nL + n \log^2 n)$	$O(\log^2 n)$	$O(L + n \log n)$
LERNA [70]	1	1 1	fwd	$O((\kappa + n)L + \kappa^2)$	$O(\kappa^2)$	$O(L)$ $O(L + n)$
SASH[72]	-	10	-	$O(L + n^2)$	-	$O(L + n^2)$
SOPA_1	-	1 1	-	$O(nL + L(\log n \log\log n))$	-	$O(L + \log nL)$ $O(nL)$
SOPA_1^p	-	1 1	-	$O(nL + L(\log n \log\log n))$	-	$O(\log nL)$ $O(n)$
SOPA_2	-	1 1	-	$O(nL + L(\log n \log\log n))$	-	$O(L + \log n)$ $O(n)$

committee members speak once too. On the contrary previous works[18, 12][1] require 8 rounds and the work of [74] 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 2). 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 [18, 72] or linearithmic [74] 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 [74] which considers a set of decryptors among the set of users.[2] Additionally, it's worth highlighting that our protocol, SOPA_2 based on the homomorphic PRG, has superior server computation complexity when compared to all other approaches.

When it comes to user communication and message sizes, previous solutions entail user complexities that either scale linearly [18, 72] or linearithmically [74] with the number of participants (n) according to Table 3. 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 in SOPA_2, 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

[1] [12] 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.

[2] Flamingo [74] 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. [74] lacks security guarantees in the event of collusion among all decryptors.

Table 3: Total received and sent asymptotic communication cost for all rounds per aggregation with semi-honest security. n denotes the total number of users, k the security parameter, L the length of the input vector, and ℓ the bit length of each element. Note that the version of our protocols relying on the LWR assumption will not incur a multiplicative k factor, as that was to represent the size of the group element being $O(k)$.

Protocol	**Communication Cost**				
	Server		User		
	Setup	Agg.	Setup	Agg.	
BIK+17[18]	-	$O(nL\ell + n^2k)$	-	$O(L\ell + nk)$	
BBG+20[12]	-	$O(nL\ell + nk\log n)$	-	$O(L\ell + k\log n)$	
Flamingo[74]	$O(k\log n)$	$O(nL\ell + nk\log^2 n)$	$O(k\log n)$	$O(L\ell + nk\log n)$	
LERNA [70]	-	$O(Lnk + \mathsf{m}^2 \cdot k)$	$O(\kappa^2 k)$	$O(\kappa L + L\log n + k)$	$O(L\kappa + L\log n + k)$
SASH [72]	-	$O(nL\ell + \kappa^2 k)$	-	$O(L\ell + nk)$	
SOPA_1	-	$O(kL(n + \log n))$	-	$O(k(L + \log nL))$	$Ok((nL + L))$
SOPA_1^p	-	$O(knL + k\log n)$	-	$O(kL + k\log n)$	$O(kn)$
SOPA_2	-	$O(knL + k\log n)$	-	$O(kL + k\log n)$	$O(kn)$

Table 4: Communication Cost, in bytes, of various SOPA protocol.

	Key-Reuse (bytes)			New Key (bytes)		
	Client→Server	Client→Per Committee	Per Committee→Server	Client→Server	Client→Per Committee	Per Committee→Server
SOPA_1	56L	56L	56L	56L	32L	32L
SOPA_1^p	56L	56	56	56L	32	32
SOPA_2	56+32L	56	56	56+32L	32	32

federated learning context which typically requires a very large number of n and L.

The works of [18, 12, 74] address an active adversary that can provide false information regarding which users are online on behalf of the server. All these works mitigate this issue 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 [18, 12] and 5 rounds in [74] with 10 rounds of setup. The setup communication complexity of [74] also increases to $O(k\log^2 n)$. This same approach applies to our protocols as well, which in turn increases just our round complexity by 2 rounds.

Communication Cost. in Table 4, we also present the communication cost of our federated learning protocols implemented using the $\mathtt{HSM}_{\mathsf{M}}$ construction. Each group element requires 56 bytes, while each field element has a size of 32 bytes. L is the length of the input vector. We present the costs for both the modes - where key-reuse mode requires the communication from client to the committee to be a group element. Meanwhile, in new key mode, the keys can be shared by itself which requires only a field element to be sent. Recall that SOPA_1^p is the packed version, which means that the client can send a single element to the committee Finally, SOPA_2 is the version of our protocol that works with a seed-homomorphic PRG.

Meanwhile, in order to ensure that malicious clients can be detected and removed, we can employ the Verifiable $One - shotPrivateAggregation$ (Construction 12), where $\mathtt{t}$ is the threshold of reconstruction. The resulting construction has the following overhead.

- For SOPA_1, it would require an additional $56Lt$ bytes from client to each committee member and from client to each server.
- For SOPA_2, this would simply be just a additional $56t$.

Note that ACORN [11] also relies on a similar Verifiable Secret Sharing approach (albeit over field

elements). However, their resulting robust protocol is non-constant round and even their non-robust version is multi-round.

Comparison with LERNA [70]. LERNA requires a fixed, stateful committee to secret share client keys, whereas we support smaller, dynamic stateless committees which 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 clients 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, SOPA_1, in the key-reuse mode, requires the client to send 5.6 MB (across both server and committee, per iteration) As a result, LERNA only becomes cost-effectively after more than 400 iterations, during which it requires a fixed, stateful, and a large committee to stay alive.

4 Outline of Paper

We begin our Technical Overview in Section 5. In Section 6, we discuss our cryptographic building blocks. In Section 6.1, we discuss the underlying cryptographic framework CL-framework [21]. We defer a discussion on class groups and cryptography to Section A. Due to space constraints, readers can refer to Section B for the syntax and security of Integer Secret Sharing. In this section, we also present the scheme from Braun *et al.* [22], while also presenting the first Packed Integer Secret Sharing scheme in the CL framework. Readers can also find the syntax and security of key-homomorphic pseudorandom functions (and its distributed variant) in Section C and a discussion of seed-homomorphic pseudorandom generators in Section E. Meanwhile, in Section 6.2, we present the first key-homomorphic PRF in the CL framework. Later, in Section 6.3 we present the distributed version of the same. Meanwhile, we defer to the appendix constructions from LWR assumption and their proofs of correctness and security to Section D.

In Section 7, we present the syntax and correctness of our primitive One-shot Private Aggregation. Due to space constraints, we present the discussion on stronger security and construction that satisfies this definition to Section 8. Meanwhile, in Section 7.3, we present the generic construction of our scheme, based on the CL framework. We present the LWR based instantiation in Sections D.2, D.3. We also present an extension to support robustness whereby a server can detect malicious behavior of clients and remove them from the aggregation. This is presented in Section F.

Our applications to Federated learning are presented in Section 9. We focus on the simpler synchronous setting in the main body of the paper while presenting the asynchronous version (i.e., a client's message to the server and committee members may be dropped) in Section H. Finally, in Section 10, we present the results from our experiments.

5 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. Our main technical tool in building this primitive

is a distributed, key homomorphic, pseudorandom function. In the following section, we present our core ideas for this construction. We begin by introducing the framework for our constructions.

CL *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 [25, 76]. The CL framework was first introduced by the work of Castagnos and Laguillaumie [33]. This framework operates on a group where there exists a subgroup with support for efficient discrete logarithm construction. Subsequent works [34, 30, 31, 94, 22] have refined the original framework. The framework has been used in various applications over the years [34, 30, 31, 98, 49, 56, 93, 22, 65]. Meanwhile, class group cryptography itself has been employed in numerous applications [96, 97, 71, 19, 32, 36, 67, 26, 6, 1, 45, 44, 7]. We use the generalized version of the framework, as presented by Bouvier *et al.* [21]. 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 $\mathtt{DCR}$ assumption. Therefore, $\mathtt{HSM}_{\mathsf{M}}$ assumption can be viewed as a generalization of the $\mathtt{DCR}$ assumption.

(Almost) Key Homomorphic Pseudorandom Functions. The earliest work that presented a key homomorphic PRF can be traced to the work of [77] where they showed that for appropriate definition of a hash function, $H(x)^k$ where k is the key and x is the input was a secure key homomorphic PRF under the DDH Assumption, in the Random Oracle Model. Subsequently, the work of [20] constructed an *almost* key homomorphic PRF under the Learning with Rounding assumption [8], again in the random oracle model. They also present constructions under the LWE assumption in the standard model. In this work, we present a new construction of key-homomorphic PRF, in the CL framework leading to new constructions under the $\mathtt{HSM}_{\mathsf{M}}$ assumption (which includes constructions based on $\mathtt{DCR}$ assumption). The construction adapts the DDH-based construction into the CL framework. We present the first such construction in the CL framework, which also includes the first such construction known from the $\mathtt{DCR}$ assumption. We formally show 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 key homomorphic PRF under the $\mathtt{HSM}_{\mathsf{M}}$ assumption.[3] The adaptation necessitates a prudent approach, one has to be careful in identifying appropriate groups to work over along with the identification of suitable input and key spaces. More precisely, while it is known that one can identify encodings of $\widehat{\mathbb{G}}$, the group order of $\mathbb{H}$ and also identification of elements in this group is not efficient. Therefore, one has to rely on using $\bar{s}$ to instantiate a distribution $\mathcal{D}_H$ such that $\{h^x : x \leftarrow\!\!\$\ \mathcal{D}_H\}$ (refer Lemma 1) is statistically indistinguishable from sampling an element directly from $\mathbb{H}$.

Distributed Key Homomorphic Pseudorandom Functions. The seminal paper of Boneh *et al.* [20] also presented generic constructions of Distributed PRF from any Key Homomorphic PRF. The reduction proceeds by secret sharing (such as using Shamir Secret Sharing [87]) the PRF key, with partial evaluation simply performing the evaluation with respect to the key share. Unfortunately, since the CL framework works in groups of unknown order, one needs to work over integer spaces. There have been several research focusing on Linear Integer Secret Sharing [47], which can be

[3] Most recently, [86] showed how to hash into groups of unknown order, such that the discrete logarithm is unknown. However, for our purposes, knowledge of discrete logarithm does not constitute an attack vector.

expensive (See an overview of shortcomings [47, Introduction]). Instead, we rely on the Shamir Secret Sharing over Integer Space, as described by Braun *et al.* [22]. This work is the first to identify how to suitably modify the secret sharing protocol to ensure that the operations can work over a group of unknown order, such as the ones we use based 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. We refine the Integer Shamir Secret Sharing scheme by adjusting various computations with appropriate offsets, to present a construction of Distributed PRFs from our $\mathtt{HSM}_{\mathsf{M}}$-based construction. Unfortunately, multiplying such offsets results in our actual evaluation protocol differing substantially from the $\mathtt{HSM}_{\mathsf{M}}$-based key homomorphic PRF. Therefore, one has to be careful during our security reduction proof to account for the offset. Finally, we show that it still satisfies the key homomorphism property whereby a combination of partial evaluations of secret shares of various keys is matched by the evaluation of the sum of the keys at the same input point.

Meanwhile, the aforementioned work of Boneh *et al.* [20] presented a generic construction of distributed PRF from any almost key homomorphic PRF. To this end, they present a construction based on the Random Oracle under the Learning with Rounding Assumption. The work of Boneh *et al.* [20] proposed the following construction $F_{\mathtt{LWR}}(\mathbf{k}, x) := \lfloor \langle \mathtt{H}(x), \mathbf{k} \rangle \rfloor_p$ as an almost key homomorphic PRF under the $\mathtt{LWR}$ assumption. Here, $q > p$ are primes where $\mathbf{k} \leftarrow\!\!\$\ \mathbb{Z}_q^\rho$ and $\mathtt{H}$ is a suitably defined hash function. Then, they present a generic reduction to build a distributed PRF from any almost key homomorphic PRF. To this end, they rely on standard Shamir's Secret Sharing over fields because both q and p are primes. This reduces some complexity when compared with integer secret sharing. However, their proposed construction contained critical shortcomings in the construction leading to issues in proof of correctness and security. We now present a broad intuition on these flaws.

An almost KH-PRF satisfies the definition that $F(k_1 + k_2, x) = F(k_1, x) + F(k_2, x) - e$ for some error e. In the case of the LWR construction, $e \in \{0, 1\}$. However, this also implies that $F(T \cdot k, x) = T \cdot F(k, x) - e_T$ where the error is now in $\{0, \ldots, T-1\}$. This results in error growth, and causes impact when we rely on Lagrange interpolation. Therefore, the fix, as proposed by the authors, is to multiply with offset Δ. By multiplying with $\Delta = \mathsf{m}!$, one can upper bound the value that the error function takes and then use suitable rounding down values to ensure that the error terms are "eaten" up. Unfortunately, in their security reduction, there comes a point where one should be able to simulate partial evaluations on i^*, such that i^* is unknown. This results in the reduction employing Lagrange interpolation with the "clearing out the denominator" technique, resulting in error growth. Therefore, the partial evaluation response again needs to be rounded down to avoid error growth. In other words, the challenger can only provide with $\left\lfloor \Delta F(k^{(i^*)}, x) \right\rfloor_u$. Recall that to round an integer down from $\mathbb{Z}_p$ to $\mathbb{Z}_u$, one finds the largest integer i such that $i \cdot \lfloor p/u \rfloor$ does not exceed x. Consequently, their definition of partial evaluation is incorrect and needs to be updated to be consistent with this simulatable response. Finally, the presence of an Δ in partial evaluation implies that the actual evaluation should have another factor of Δ. We also noticed further issues in their choices of rounding. Specifically, the partial evaluation needs to be rounded down to u such that $\lfloor p/u \rfloor > ((\Delta + 1) \cdot \mathsf{t} \cdot \Delta)$ where t is the threshold for reconstruction. Additionally, they only provided a framework to build distributed PRF from any almost key homomorphic PRF with respect to one PRF key and not the case where the key is a vector, as is the case with LWR. We fix these issues

in our construction where we build a distributed, almost key homomorphic PRF from the LWR-based almost key homomorphic PRF. We formally show that $F_{\mathtt{LWR}}(\mathbf{k}, x) = \left\lfloor \Delta \left\lfloor \Delta \left\lfloor \langle \mathtt{H}(x), \mathbf{k} \rangle_p \right\rfloor_u \right\rfloor \right]_v$ for appropriate choices of u and v is a secure, distributed, almost key homomorphic, PRF.

One-shot Private Aggregation With these key cryptographic building blocks in place, we introduce a new primitive called One-shot Private Aggregationfor secure aggregation for multiple iterations where users choose to participate in a particular iteration or not. The idea of this primitive is for the client to do two steps concurrently: encrypt its input value while providing the committee members with some auxiliary information. Meanwhile, the committee members can combine the auxiliary information from all of the clients and forward the results to the server. Now, the server can accumulate all of the ciphertexts, use the information from the committee members, and finally unmask the summation. We build generic constructions from any distributed, key homomorphic PRF and prove its security. The idea of our construction is as follows. Client i, with input $x_{i,\ell}$ for iteration ℓ, first uses its PRF key k_i to compute $F(k_i, \ell)$ and uses the result to mask $x_{i,\ell}$. In the CL framework, we do this as follows: $f^{x_{i,\ell}} \cdot F(k_i, \ell)$. Meanwhile, to the committee member j, it sends $F(k_i^{(j)}, \ell)$ where $k_i^{(j)}$ is the j-th share of using Integer Shamir Secret Sharing to distribute the key k_i. The committee member j, which has received $\left\{F(k_i^{(j)}, \ell)\right\}_{i=1}^{n}$ simply combines the inputs it received to obtain $F(\sum_{i=1}^{n} k_i^{(j)}, \ell)$ using the key-homomorphism property. This is communicated to the server. Once $\mathtt{t}$ such combinations are received by the server, the server uses the reconstruction property of the integer secret sharing scheme to compute $F(\sum_{i=1}^{n} k_i, \ell)$. Further, it multiplies all the ciphertexts together to get $f^{\sum x_{i,\ell}} \cdot \prod_{i=1}^{n} F(k_i, \ell)$. Finally, it can use the key homomorphism property again to remove the product mask and simply get $f^{\sum x_{i,\ell}}$, which it can then recover because $\mathbb{F}$ has efficient discrete logarithm. The CL framework also supports the construction from the DCR assumption.

Additionally, we also build OPA from the distributed, almost key homomorphic PRF based on LWR. However, due to the "almost key homomorphism", the error growth required additional rounding. Therefore, we build our OPA based on $\mathtt{LWR}$ assumption, in a white-box manner. In our construction, the auxiliary information only consists of the first level of rounding down. Specifically, $\mathsf{aux}_{i,\ell}^{(j)} = \left\lfloor \langle \mathtt{H}(\ell \rangle, \mathbf{k}_i^{(j)} \right\rfloor_p$ where $\mathsf{aux}_{i,\ell}^{(j)}$ is the information sent from client i to committee member j for label ℓ with $\mathbf{k}_i^{(j)}$ denoting the j-th share of $\mathbf{k}_i$ where the sharing was done using Shamir's Secret Sharing over a field. Meanwhile, the committee member simply sums up all of the auxiliary information it has received, and multiplies it with the offset before rounding down to an appropriate choice of integer u to avoid any error growth. In essence, the output at this stage corresponds with a partial evaluation of x with the j-th key share corresponding to $\sum_{i=1}^{n} \mathbf{k}_i$. Finally, the server combines it using the standard DPRF to recover aux_ℓ which corresponds with $F(\sum_{i=1}^{n} \mathbf{k}_i, x)$ based on the correctness of our DPRF. Additional care needs to be taken to ensure that the input $x_{i,\ell}$ is suitably modified before encrypting to ensure that the aggregate can be computed, even in the face of leakage.

However, the resulting construction still has worse parameters than desired due to successive rounding. We, instead, present a more efficient construction, under the $\mathtt{LWR}$ assumption with the following observation: Unlike other protocols in secure aggregation literature, in our case, one does not need to reuse the key for correctness. This leverages the non-interactivity critically. Indeed, while reusing keys does seem to naively offer computational efficiency, we show that for our $\mathtt{LWR}$ setting, the flexibility afforded by keys per label is far higher. We employ a key $\mathbf{k}_{i,\ell}$ for each user i for each label ℓ. The benefits are listed below:

- Reusing keys necessitated that the auxiliary information never leaked information about the

key share, while still allowing computation. Meanwhile, since we have a key per label, and each label is used exactly once, the shares received by the committee can simply be the key share.

- Since the key shares are provided only by the client, rather than associating a separate sharing polynomial per element of the key vector, one can resort to packed secret sharing. This reduces the communication sent from the client to any one committee member to simply be a *single* field element.
- This savings can be forwarded to the server as well with the committee members simply naively adding its received key shares and sending the information to the server. Then, the server can perform the Lagrange Interpolation to receive the aggregated key, element by element.
- Finally, it can use the almost key homomorphism of the PRF to unmask the aggregated inputs.

Stronger Security Definitions and Construction. We also present a stronger security guarantee 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 own 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.

OPA *and Private Stream Aggregation.* Private Stream Aggregation was a primitive introduced to ensure that a server can routinely aggregate information from the clients without learning any information about the client's inputs. There has been a long line of research in this domain which has worked under the setting where a trusted setup distributes the key. Meanwhile, the only work that aimed to allow for dynamic participation while avoiding the pitfalls of trusted setup focused on a modified setting of private stream aggregation whereby it used a "collector" [68]. This implies that a party cannot drop or join, without redoing the entire setup. Meanwhile, our OPA protocol can be effectively used as a drop-in for Private Stream Aggregation protocols. It allows dynamic participation where users can join and leave, as needed. We have distributed trust among the committee members. It also avoids some issues of having to compute expensive operations to recover the aggregate (i.e., take discrete logarithm). Finally, we also present a version of PSA that is robust, i.e., incorrect client behavior can be detected and removed, along with any errant behavior on the part of the committee members. More details can be found in Section G.

Secure Aggregation for Federated Learning. Using One-shot Private Aggregation we present three levels of construction.

- SOPA_1: To begin with, we present a naive approach whereby our OPA primitive is used to encrypt a vector of length L, element by element. This requires a key per element of the

vector. The server receives L ciphertexts, per client while the committee members receive L key shares per client. In this protocol, which we call SOPA_1, we consider the synchronous setting in which all the communication sent by the set $\mathcal{C}$ of clients participating in an iteration reaches the designated parties. Then, one can simply use our OPA protocol in a black box fashion, applied to every element separately to ensure that the server is capable of aggregating the clients' vector of inputs. For simplicity, we assume that $\mathsf{m} = \log n$. However, our protocol can work for any choice of m.

- We propose an optimization to use packed secret sharing. Notice that there are L keys that were originally distinctly shared using several polynomials. This can be optimized by using a single polynomial to share all these L secrets, reducing the communication between the client and the committee members to a minimum. Note that one requires Secret Sharing over the Integer Space to ensure that we can pack secret shared multiple instances of our OPA construction based on the $\mathtt{HSM}_{\mathsf{M}}$ assumption. We show how to adapt existing Shamir's Packed Secret Sharing protocol over prime-order fields to integers in this work. Our proof technique again relies on constructing a vector of sweeping polynomials. We believe this is a contribution of independent interest. We denote the packed, grouped version of our protocol with SOPA_1^p.
- SOPA_2: Observe that, in SOPA_1, we encrypt each element separately and if one were to use the $\mathtt{HSM}_{\mathsf{M}}$ assumption, this can prove to involve $\Omega(L)$ group exponentiations which can be impractical when L is very large. Instead, we propose a hybrid approach to reduce communication and computation. The solution relies on the approach observed by SASH [72]. The idea is to use a seed homomorphic PRG, i.e., we have a PRG $\mathsf{SHPRG} : \mathbb{Z}_p \to \mathbb{Z}_p^L$ $\mathsf{SHPRG}(seed_1) + \mathsf{SHPRG}(seed_2) = \mathsf{SHPRG}(seed_1 + seed_2)$. With this tool in our arsenal, we can do this simpler solution. Each client i samples a seed $seed_{i,\ell}$ per iteration ℓ. Then, the client i uses our OPA protocol to encrypt $seed_{i,\ell}$. At the end of the execution of our OPA protocol for iteration ℓ, the server receives $\sum_{i=1}^{n} seed_{i,\ell}$. Meanwhile, the client uses SHPRG to expand $seed_{i,\ell}$ to get $(mask_{i,\ell}^{(1)}, \ldots, mask_{i,\ell}^{(L)}) \leftarrow \mathsf{SHPRG}(seed_{i,\ell})$. Then, the client can mask the actual input vector by doing $x_{i,\ell}^{(j)} + mask_{i,\ell}^{(j)}$ for $j = 1, \ldots, L$. Finally, the server can use the aggregated seed produced by our OPA protocol to compute the sum of all the masks and then unmask it to receive the aggregation. Unfortunately, there is only a construction based on LWR that is almost seed homomorphic [20], so care must be taken to handle the error. Critically, this reduces the actual encryption of the input vector to operations over field elements. Moreover, the communication with the committee members is reduced as there is only a single key to share. Finally, of independent interest, we also present a seed homomorphic PRG from $\mathtt{HSM}_{\mathsf{M}}$ assumption.
- SOPA_3: SOPA_3 that aims to handle the case when the set of clients whose inputs are received by the server is different from the committee members. While the client still speaks once, we add two rounds of communication between the committee members and the server to identify the set $\mathcal{C}$ of users whose messages have been received by the server and a sufficient quorum of committee members.

6 Cryptographic Building Blocks

We defer our exposition on some of our cryptographic building blocks to our appendix. We discuss class groups and cryptography in Section A. For completeness, we discuss secret sharing in Section B. We discuss pseudorandom functions in Section C.

6.1 Generalized CL Framework

Let M be an integer. The CL Framework consists of a setup algorithm that generates a large cyclic group of an unknown order but which contains a subgroup of order M where the discrete

logarithm problem is easy. The index of that subgroup is dictated by some security parameter of the scheme. Consequently, one can admit various versions of the Elgamal cryptosystem [51] such that the plaintext space is the additive group $(\mathbb{Z}/\mathsf{M}\mathbb{Z}, +)$. While the work of Castagnos and Laguillaumie [33] and its subsequent variations and refinement restricted the choice of M to be a prime integer M, the work of Bouvier *et al.* [21] presented a more generic framework that would capture an assortment of linearly homomorphic encryption schemes. Specifically, the generic framework implies the security of the construction where $\mathsf{M} = p$ [34], $\mathsf{M} = p^k$ for some choice of k [48], where $\mathsf{M} = 2^k$ [35]. They also show that it encompasses schemes which go beyond class groups including variants of the Paillier cryptosystem [79] such as the Camenisch and Shoup's encryption scheme [29] where $\mathsf{M} = N$ which is an RSA modulus. At it core lies the Hidden Subgroup Membership ($\mathtt{HSM}_{\mathsf{M}}$) assumption which can be viewed as a generalization of the Decisional Composite Residuosity ($\mathtt{DCR}$) assumption.

Definition 1 (Generalized CL Framework)**.** *Let κ_s be the statistical security parameter and let κ_c be the computational security parameter. Let M be a positive integer. Let $\widehat{\mathbb{G}}$ be a group of unknown order $\mathsf{M} \cdot \hat{s}$, for some $\hat{s}$ that depends on the security parameter κ_c. We require that $\widehat{\mathbb{G}}$ admits a cyclic group $\mathbb{F}$, generated by f, of order M such that $\gcd(\mathsf{M}, \hat{s}) = 1$, although $\hat{s}$ which is the index $\widehat{\mathbb{G}} : \mathbb{F}$ is unknown and hard to compute. Instead, we are given an upper bound $\bar{s}$ of $\hat{s}$. Further, let $h = x^{\mathsf{M}}$ for some random $x \in \widehat{\mathbb{G}}$.*

The framework is defined by two algorithms $(\mathsf{CLGen}, \mathsf{CLSolve})$ such that:

- $\mathsf{pp}_{\mathsf{CL}} = (\bar{s}, f, h, \widehat{\mathbb{G}}, \mathbb{F}) \leftarrow\!\!\$\ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s})$[4]
- *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 $\mathbb{F}$:*

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

Proposition 1 ([21])**.** *Let $g := f \cdot h$. Let $\mathbb{G} := \langle g \rangle$ be a cyclic subgroup of $\widehat{\mathbb{G}}$. If $\mathbb{H} = \{x^M : x \in \mathbb{G}\}$, then we have $\mathbb{H}$ is generated by h and $\mathbb{G}$ factors as $\mathbb{F} \times \mathbb{H}$.*

Definition 2 (Intermediate Distributions)**.** *For ease in security proofs, we will define intermediate distributions as follows. $\mathcal{D}_G$ (resp. $\mathcal{D}_H$) be a distribution over the set of integers such that the distribution $\{g^x : x \leftarrow\!\!\$\ \mathcal{D}_G\}$ (resp. $\{h^x : x \leftarrow\!\!\$\ \mathcal{D}_H\}$) is at most distance $2^{-\kappa_s}$ from the uniform distribution over $\mathbb{G}$ (resp. $\mathbb{H}$).*

We also have the following lemma from Castagnos, Imbert, and Laguillaumie [32] which defines how to sample $\mathcal{D}_G, \mathcal{D}_H$ from a discrete Gaussian distribution.

Lemma 1. *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:*

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

[4] In the original work where $\mathsf{M} = p$, the CLGen algorithm was also allowed to take as input the M with the only requirement that it was at least κ_c bits long and the randomness used in the procedure was made public. This implies that one does not need to rely on a trusted setup. These are features we will later exploit to make this entire CLGen algorithm be decentralized, as shown by the work of Castagnos *et al.* [30, 31].

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 1, we get that the distribution is less than $2^{-\kappa_s}$ away from uniform distribution in $\mathbb{H}$. We refer the readers to the discussion in Tucker [94, §2.7.1, §3.1.3, §3.7] for more information about instantiating the distribution from a folded uniform distribution or from a folded discrete Gaussian distribution.

Definition 3 (Hard Subgroup Membership Assumption ($\mathtt{HSM}_{\mathsf{M}}$) Assumption [34])**.** *Let κ_c be a the security parameter. Let M be a positive integer. Let $(\mathsf{CLGen}, \mathsf{CLSolve})$ be the group generator algorithms as defined in Definition 1, then the $\mathit{HSM}_{\mathsf{M}}$ assumption requires that that $\mathit{HSM}_{\mathsf{M}}$ problem be hard in $\mathbb{G}$ even with access to the Solve algorithm. More formally, we say that the $\mathit{HSM}_{\mathsf{M}}$ problem is hard if for all PPT adversaries $\mathcal{A}$, there exists a negligible function* negl *such that:*

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathsf{pp}_{\mathsf{CL}} \leftarrow\!\!\$\ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s}) \\ x \leftarrow\!\!\$\ \mathcal{D}_H, u \leftarrow\!\!\$\ (\mathbb{Z}/\mathsf{M}\mathbb{Z}) \backslash \{0\} \\ b \leftarrow\!\!\$\ \{0,1\}; Z_0 = h^x; Z_1 = f^u \cdot h^x \\ b' \leftarrow\!\!\$\ \mathcal{A}^{\mathsf{Solve}(\mathsf{pp}_{\mathsf{CL}}, \cdot)}(\mathsf{pp}_{\mathsf{CL}}, Z_b) \end{array} \right] \leqslant \frac{1}{2} + \mathrm{negl}(\kappa_c)$$

Remark 1 (Trusted Setup)**.** It is well known that the $\mathtt{DCR}$ assumption requires a trusted setup where the RSA modulus N is the product of two sufficiently large safe primes. Any knowledge of the independent primes renders the assumption insecure. As pointed out by the work of Bouvier *et al.* [21], the procedure for CLGen for the case when $\mathsf{M} = 2^k$ also requires an RSA modulus. Consequently, when $\mathsf{M} = q, q^k$, one does not require a trusted setup while $\mathsf{M} = N$ and $\mathsf{M} = 2^k$ where N is an RSA modulus that requires a trusted setup. We refer the interested readers to the work of Bouvier *et al.* [21, §4] for a detailed exposition on how to instantiate the cases for $\mathsf{M} = q^k, 2^k$ in class groups.

6.2 Pseudorandom Functions in CL Framework

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

Remark 2. Note that the order of $\mathbb{H}$ is unknown. Therefore, one has to rely on $\mathcal{D}_H$ to hash into $\mathbb{H}$. Most recently, [86] 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 2. *Construction 1 is a secure PRF where $\mathtt{H}$ is modeled as a random oracle under the $\mathit{HSM}_{\mathsf{M}}$ 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 [41].

$\mathsf{Hybrid}_0(\kappa)$**:** Corresponds to the security game as defined in Definition 10. It follows that the advantage of the adversary is

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

$\mathsf{Hybrid}_1(\kappa)$**:** 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,

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

□

Remark 3. 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.* [77]. It is easy to verify that our construction is also key homomorphic as $H(x)^{(k_1+k_2)} = H(x)^{k_1} \cdot H(x)^{k_2}$.

6.3 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 Construction 4.

Construction 2 (Distributed PRF in CL Framework)**.** 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 1.

Theorem 3. *In the Random Oracle Model, if Construction 1 is a secure pseudorandom function if Construction 4 is statistically private, then Construction 2 is pseudorandom in the static corruptions setting.*

Correctness. As discussed in Section B.1, 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 $f(X) := \sum_{j=1}^{\mathsf{t}} \lambda_{i_j}(X) \cdot \mathsf{k}^{(i_j)}$.

Protocol Distributed PRF

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

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 $\mathsf{pp}_\mathsf{PRF} := (\mathsf{pp}_\mathsf{CL}, \mathsf{LISS.pp}_\mathsf{SS}, \mathtt{H})$

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

Run $\mathsf{k}^{(1)}, \ldots, \mathsf{k}^{(\mathsf{m})} \leftarrow\!\!\$\ \mathsf{LISS.Share}(\mathsf{k}, \mathsf{t}, \mathsf{m})$
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 $\Lambda_{i \in \mathcal{S}} = \mathsf{LISS.GetCoeff}(\mathcal{S})$
Compute $Y' = \prod_{i \in \mathcal{S}} y_i^{\Lambda_i}$
return Y'

Figure 1: Construction of Distributed PRF based on the $\mathsf{LISS} := (\mathsf{Share}, \mathsf{GetCoeff}, \mathsf{Reconstruct})$ scheme of Construction 4, with pp_SS denoting the public parameters of the LISS scheme. Recall that the offset $\Delta := \mathsf{m}!$ where m is the number of shares generated.

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.

Pseudorandomness. Next, we consider the pseudorandomness property of our construction. Boneh *et al.* [20] showed that from any Key Homomorphic PRF (which Construction 1), 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 1. The idea behind the proof is for $\mathcal{B}$, upon receiving a 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 knows 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 P-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 1. 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. □

Verification of Key Homomorphism. Let k_1 and k_2 be two sampled keys from $\mathcal{K}$. Let $\mathsf{k}_1^{(1)}, \ldots, \mathsf{k}_1^{(\mathsf{m})} \leftarrow\!\!\$\ \mathsf{Share}(\mathsf{k}_1)$ and $\mathsf{k}_2^{(1)}, \ldots, \mathsf{k}_2^{(\mathsf{m})} \leftarrow\!\!\$\ \mathsf{Share}(\mathsf{k}_2)$ where we have two polynomials $f_1(X)$ and $f_2(X)$ with the property such that $f_1(X) = \Delta \cdot \mathsf{k}_1, f_2(X) = \Delta \cdot \mathsf{k}_2$. Consider a subset $\mathcal{S}$ of size at least t indicated by the indices $\{i_1, \ldots, i_\mathsf{t}\}$. Then, the lagrange coefficients induced by the set $\mathcal{S}$ can be defined as: $\forall\ i_j \in \mathcal{S}, \lambda_{i_j}(X) := \prod_{\zeta \in [\mathsf{t}] \setminus \{i\}} \frac{i_\zeta - X}{i_\zeta - i_j}$. This implies that the resulting polynomial is $f_1(X) := \sum_{j=1}^{\mathsf{t}} \lambda_{i_j}(X) \cdot \mathsf{k}_1^{(i_j)}, f_2(X) := \sum_{j=1}^{\mathsf{t}} \lambda_{i_j}(X) \cdot \mathsf{k}_2^{(i_j)}$. Similarly, as before we will consider $\Lambda_i(X) := \Delta \cdot \lambda_i(x)$.

Thereby, the resulting polynomials are $f_1'(X) = \Delta \cdot f_1(X) = \sum_{j=1}^{\mathsf{t}} \Lambda_{i_j}(X) \cdot \mathsf{k}_1^{(i_j)}, f_2'(X) = \Delta \cdot f_2(X) = \sum_{j=1}^{\mathsf{t}} \Lambda_{i_j}(X) \cdot \mathsf{k}_2^{(i_j)}$.

Then, for any x, consider $y_{1,2}^{(j)} := \mathtt{H}(x)^{\Delta \mathsf{k}_1^{(j)}} \cdot \mathtt{H}(x)^{\Delta \mathsf{k}_2^{(j)}} = \mathtt{H}(x)^{\Delta \cdot (\mathsf{k}_1^{(i_j)} + \mathsf{k}_2^{(i_j)})}$ for $j = 1, \ldots, \mathsf{m}$. Then, let us consider:

$$\begin{aligned}\mathsf{Eval}(\mathsf{k}_1 + \mathsf{k}_2, x) := \mathtt{H}(x)^{\Delta^3(\mathsf{k}_1 + \mathsf{k}_2)} &= \mathtt{H}(x)^{\Delta^3 \mathsf{k}_1} \cdot \mathtt{H}(x)^{\Delta^3 \mathsf{k}_2} = \mathtt{H}(x)^{\Delta^2 \cdot f_1(0)} \mathtt{H}(x)^{\Delta^2 f_2(0)} \\ &= \mathtt{H}(x)^{\Delta \cdot f_1'(0) + f_2'(0)} = \mathtt{H}(x)^{\Delta \cdot \sum_{j=1}^{\mathsf{t}} \Lambda_{i_j}(0) \cdot (\mathsf{k}_1^{(i_j)} + \mathsf{k}_2^{(i_j)})} \\ &= \left(\prod_{j=1}^{\mathsf{t}} \mathtt{H}(x)^{\Delta \cdot (\mathsf{k}_1^{(i_j)} + \mathsf{k}_2^{(i_j)})}\right)^{\Lambda_{i_j}(0)} = \left(\prod_{j=1}^{\mathsf{t}} y_{1,2}^{(i_j)}\right)^{\Lambda_{i_j}(0)} \\ &= \mathsf{Combine}(\left\{y_{1,2}^{(i_j)}\right\}_{j \in [\mathsf{t}]}\end{aligned}$$

As discussed earlier, the $\mathtt{HSM}_\mathsf{M}$ assumption also generalizes the $\mathtt{DCR}$ assumption. It follows that we also have a Distributed PRF that is Key Homomorphic under the DCR Assumption in the Random Oracle model.

Corollary 4. *In the Random Oracle model, there exists a secure distributed, key homomorphic PRF evaluating the PRF $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$*

We defer our constructions, security, and proofs of correctness from $\mathtt{LWR}$ assumption to Section D.1.

7 One-shot Private Aggregation

In this section, we begin by introducing the syntax and security of our primitive which we call (Labeled) 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.

7.1 Syntax and Correctness

Definition 4. *Let* $C = \{C_1, \ldots, C_{\mathsf{m}}\}$ *be a committee of size* m *and let* t *be the threshold for the committee. Further, let* M *denote the modulus of the summation and* $\mathcal{L}$ *be the set of all labels. Then, an* $(\mathsf{t}, \mathsf{m}, \mathsf{M})$ *One-shot Private Aggregation Scheme* $\mathsf{OPA} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{C\text{-}Combine}, \mathsf{S\text{-}Combine}, \mathsf{Aggregate})$ *consists of five algorithms and has the following syntax:*

- $\mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa, 1^{\mathsf{t}}, 1^{\mathsf{m}})$*: On input of a security parameter* κ*, public parameters* pp *are generated. These parameters are implicit arguments to the remaining algorithms.*
- $\mathsf{k}_i \leftarrow\!\!\$\ \mathsf{KeyGen}()$*: Generates and outputs a party's private key* k_i*. This algorithm is run by the server and the clients, but not the committee.*
- $\left(\mathsf{ct}_{i,\ell}, \left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in[\mathsf{m}]}\right) \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pp}, \mathsf{k}_i, x_i, \mathsf{t}, \mathsf{m}, \ell)$*: For label* $\ell \in \mathcal{L}$*, client* i *encrypts its value* x_i *using its private key* k_i *and produces* $\mathsf{ct}_{i,\ell}$*. Further, it also generates an auxiliary information for label* ℓ*,* $\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in[\mathsf{m}]}$ *where* $\mathsf{aux}_{i,\ell}^{(j)}$ *is sent to committee member* j*.*
- $(\mathsf{AUX}_\ell^{(j)}) \leftarrow \mathsf{C\text{-}Combine}(\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{i\in\mathcal{C}_\ell})$*: This is the "combine" procedure that is executed by the committee member, whereby a member* j *runs this algorithm on all the auxiliary information* $\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{i\in\mathcal{C}_\ell}$ *for label* ℓ *it has received from the clients denoted by* $\mathcal{C}_\ell$*, and outputs the combination of this auxiliary input* $\mathsf{AUX}_\ell^{(j)}$*.*
- $\mathsf{AUX}_\ell \leftarrow \mathsf{S\text{-}Combine}(\left\{\mathcal{C}, \mathsf{AUX}_\ell^{(j)}\right\}_{j\in\mathcal{S}}))$*: This is the "combine" procedure that is executed by the server, on the combined information received from a set* $\mathcal{S}$ *of committee members, denoted by* $\mathcal{S} \subseteq \{1, \ldots, m\}$ *such that* $|\mathcal{S}| \geqslant \mathsf{t}$ *where the combined information is denoted by* $\left\{(\mathsf{AUX}_\ell^{(j)})\right\}_{j\in\mathcal{S}}$*, and outputs the final combined information for* ℓ *denoted by* AUX_ℓ*.*
- $X_\ell \leftarrow \mathsf{Aggregate}(\mathsf{AUX}_\ell, \{\mathsf{ct}_{i,\ell}\}_{i\in\mathcal{C}})$*: The server, on input of* AUX_ℓ*, the ciphertexts* $\{\mathsf{ct}_{i,\ell}\}_{i\in\mathcal{C}}$*, runs the aggregate procedure to output* X_ℓ *such that* $X_\ell = \sum_{i\in\mathcal{C}} x_i \bmod M$*.*

Definition 5 (Correctness)**.** *We say that a* $(\mathsf{t}, \mathsf{m}, \mathsf{M})$ *One-shot Private Aggregation Scheme* OPA *is correct if: for any* $n, \mathsf{m}, \kappa \in \mathbb{Z}$*, any* $\mathsf{t} \in \mathbb{Z}$ *such that* $\mathsf{t} < \mathsf{m}$*, any set* $\mathcal{S} \subseteq [\mathsf{m}]$ *with* $|\mathcal{S}| \geqslant \mathsf{t}$*, any inputs* $x_1, \ldots, x_n \in \mathbb{Z}_{\mathsf{M}}$ *with* $X'_\ell := \sum_{i\in[n]} x_i \bmod \mathsf{M}$*, and any label* $\ell \in \mathcal{L}$*, the following holds:*

$$\Pr\left[\begin{array}{c|c} X_\ell = X'_\ell & \begin{array}{c} \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa), \{\mathsf{k}_i \leftarrow\!\!\$\ \mathsf{KeyGen}()\}_{i\in[n]\cup\{0\}} \\ \left\{\left(\mathsf{ct}_{i,\ell}, \left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in[\mathsf{m}]}\right) \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pp}, \mathsf{k}_i, x_i, \mathsf{t}, \mathsf{m}, \ell)\right\}_{i\in[n]} \\ \left\{(\mathsf{AUX}_\ell^{(j)}) \leftarrow \mathsf{C\text{-}Combine}\left(\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{i\in[n]}\right)\right\}_{j\in\mathcal{S}} \\ \mathsf{AUX}_\ell \leftarrow \mathsf{S\text{-}Combine}(\left\{\mathsf{AUX}_\ell^{(j)}\right\}_{j\in[\mathcal{S}]}) \\ X_\ell \leftarrow \mathsf{Aggregate}\left(\mathsf{AUX}_\ell, \{\mathsf{ct}_{i,\ell}\}_{i\in[n]}\right) \end{array} \end{array}\right] = 1$$

Remark 4. The above syntax is for the "Key Reuse" mode. For the new key mode, the KeyGen algorithm will output $\mathsf{k}_{i,\ell}$

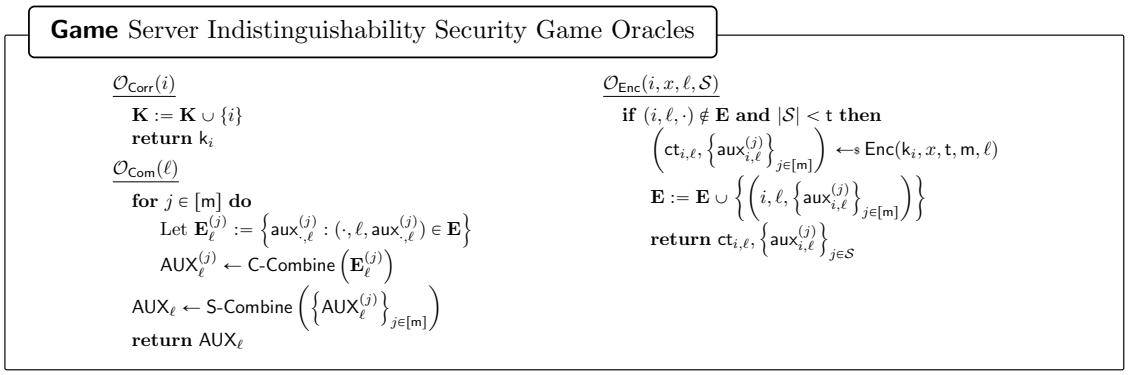

Figure 2: Server Indistinguishability Security Game Oracles

7.2 Server Indistinguishability

The server indistinguishability game aims to guarantee that the server cannot learn the input of any honest party, even if it receives up to $\mathsf{t}-1$ auxiliary information sent by the client and all the collected auxiliary information produced by all the committee members. This is modeled by allowing the adversary to make queries to various oracles, as defined in Figure 2, which we now expand upon in words. For convenience, we introduce an intermediate function $\mathsf{S\text{-}Combine}$ that we employ in the security game:

- Corrupt the user ($\mathcal{O}_{\mathsf{Corr}}$): This is the oracle that the adversary uses to corrupt and retrieve the secret key corresponding to a user i that it specifies. It receives k_i in response. We use $\mathbf{K}$ to record all corrupted parties.
- Encrypt, on behalf of a user ($\mathcal{O}_{\mathsf{Enc}}$): The adversary invokes this oracle with the following inputs: the identity of the user i, the input to be encrypted x, the label to be encrypted under ℓ, and the committee subset $\mathcal{S}$ for which it wishes to receive the auxiliary information. We record $(i, \ell, \left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in[\mathsf{m}]})$ in $\mathbf{E}$. There are two restrictions in making queries to this oracle. First, the adversary can only query this oracle once per combination of (i, ℓ) and we use $\mathbf{E}$ to do this verification. Second, the size of $\mathcal{S}$ does not exceed the threshold, i.e, $|\mathcal{S}| < \mathsf{t}$. In response, the adversary receives the ciphertext that encrypted the input $x_{i,\ell}$ and auxiliary information for each of the members in $\mathcal{S}$, i.e., $\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in\mathcal{S}}$.
- Combination queries, on the inputs that have been encrypted to label ℓ ($\mathcal{O}_{\mathsf{Com}}$): The adversary invokes this oracle to receive the final combined auxiliary information AUX_ℓ where the auxiliary information is a function of all those auxiliary information generated through calls to $\mathcal{O}_{\mathsf{Enc}}$ on adversarial choices of input.

The server indistinguishability game proceeds in phases.

- Setup Phase: The challenger runs the setup algorithm to generate the system parameters pp. The adversary is given pp and outputs n, the number of users to register. The challenger runs the KeyGen algorithm n times, once for each of the n users.
- Learning Phase: The adversary issues queries to the various oracles defined by $\mathcal{O}_{\mathsf{Corr}}, \mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Com}}$ to learn any information it could. This phase ends with the adversary committing to a target label τ.

- Challenge Phase: In this phase, the challenger identifies honest eligible users $\mathcal{U} := [n] \backslash \mathbf{K}$. 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 a 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, with the caveat that the sum needs to be the same for those in $\mathcal{H}$.
- Finally, the adversary is provided with individual auxiliary information for the subset $\mathcal{S}$ of the committee members, ciphertexts, and the final auxiliary information for τ denoted by AUX_τ.
- Guessing Phase: The adversary outputs a guess b' and wins if $b' = b$, provided trivial attacks do not happen.

Definition 6 (S-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:*

$$\Pr\left[\begin{array}{c|c} \mathsf{Check}(\{x_{i,0}, x_{i,1}\}_{i \in \mathcal{H}}) \wedge & \mathsf{pp} \leftarrow\!\!\$\ \mathsf{Setup}(1^\kappa); b \leftarrow\!\!\$\ \{0,1\} \\ b = b' & (st, n) \leftarrow\!\!\$\ \mathcal{A}(\mathsf{pp}), \{\mathsf{k}_i \leftarrow\!\!\$\ \mathsf{KeyGen}()\}_{i \in [n]} \\ |\mathcal{S}| < \mathsf{t} & (st, \tau) \leftarrow\!\!\$\ \mathcal{A}^{\mathcal{O}_{\mathsf{Corr}}, \mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Com}}}(st), \mathcal{U} := [n] \backslash \mathbf{K} \\ & (\mathcal{H}, \mathcal{S}, \{x_{i,0}, x_{i,1}\}_{i \in \mathcal{H}}, \{x_i\}_{i \in [n] \backslash \mathcal{H}}) \leftarrow\!\!\$\ \mathcal{A}(st, \mathcal{U}) \\ & \left\{\mathsf{ct}_{i,\tau}, \left\{\mathsf{aux}_{i,\tau}^{(j)}\right\}_{j \in [\mathsf{m}]} \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pp}, r_i, x_{i,b})\right\}_{i \in \mathcal{H}} \\ & \left\{\mathsf{ct}_{i,\tau}, \left\{\mathsf{aux}_{i,\tau}^{(j)}\right\}_{j \in [\mathsf{m}]} \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{pp}, r_i, x_i)\right\}_{i \in [n] \backslash \mathcal{H}} \\ & \left\{(\mathsf{AUX}_\tau^{(j)}) \leftarrow \mathsf{C\text{-}Combine}\left(\left\{\mathsf{aux}_{i,\tau}^{(j)}\right\}_{i \in [n]}\right)\right\}_{j \in [\mathsf{m}]} \\ & \mathsf{AUX}_\tau \leftarrow \mathsf{S\text{-}Combine}(\left\{(\mathsf{AUX}_\tau^{(j)}\right\}_{j \in [\mathsf{m}]} \\ & b' \leftarrow\!\!\$\ \mathcal{A}(st, \left\{\mathsf{ct}_{i,\tau}, \mathsf{aux}_{i,\tau}^{(j)}\right\}_{i \in [n], j \in \mathcal{S}}, \mathsf{AUX}_\tau) \end{array}\right] \leqslant \frac{1}{2} + \mathrm{negl}(\kappa)$$

where

$\mathsf{Check}(\{x_{i,0}, x_{i,1}\}_{i \in \mathcal{H}})$
  if $\sum_{i \in \mathcal{H}} x_{i,0} \bmod M \neq \sum_{i \in \mathcal{H}} x_{i,1} \bmod M$ **then**
    return false
  return true

7.3 Our Construction of One-shot Private Aggregation Scheme

In this section, we present our constructions of OPA and prove its correctness and security. While our underlying abstraction is that we build it from a distributed, key-homomorphic PRF $\mathsf{DPRF} := (\mathsf{DPRF.Gen}, \mathsf{DPRF.Share}, \mathsf{DPRF.Eval}, \mathsf{DPRF.P\text{-}Eval}, \mathsf{DPRF.Combine})$ as defined in Definition 12, we would like to point out that these schemes need to have a particular feature. Specifically, if the range of our PRF is $\mathcal{Y}$. Then, there exists a subgroup $\mathcal{W}$ into which our desired inputs can be efficiently encoded and decoded. In our $\mathtt{HSM}_{\mathsf{M}}$ based construction, the subgroup is $\mathbb{F}$ from where one can efficiently encode and decode an input in $\mathbb{Z}/\mathsf{M}\mathbb{Z}$. This is much easier in the context of our $\mathtt{LWR}$ construction as everything happens over subgroups of the integers.

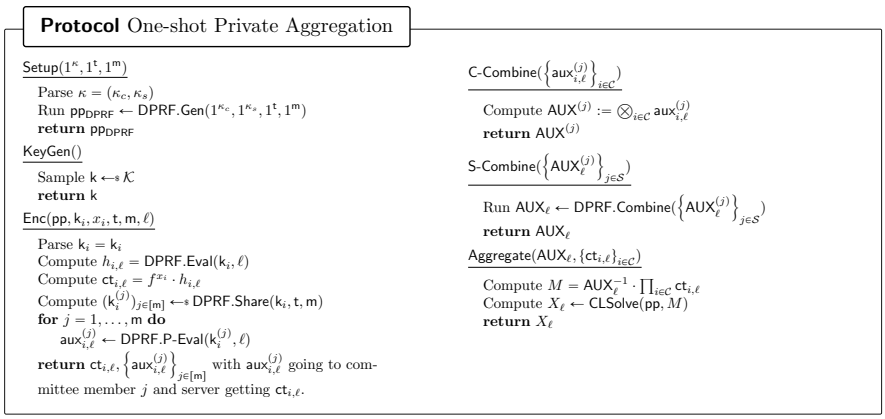

Figure 3: Our Construction of OPA based on the $\mathtt{HSM}_{\mathsf{M}}$ Assumption.

We present the construction that builds OPA from a KH-DPRF as instantiated from the $\mathtt{HSM}_{\mathsf{M}}$ Assumption. Due to space constraints, we defer our LWR constructions to the appendix. We present a construction that builds OPA from an almost KH-TPRF, as instantiated from the LWR Assumption in Section D.2. We also present a much simpler construction in Section D.3 whereby there is a new key in every round/label, i.e., in the "New Key" mode. We present this version to have much better parameters based on the LWR assumption. We maintain that the construction based on Distributed Key Homomorphic PRF achieves a stronger security definition whereby, even if the same key is used across multiple labels, it does not affect the security of the construction. However, for some applications, the optimized version is sufficient whereby a new key is generated at every round.

We build OPA based on the $\mathtt{HSM}_{\mathsf{M}}$ Assumption, building it based on the Key Homomorphic, Distributed PRF as presented in Construction 1.

Construction 3. We present our construction in Figure 3.

Correctness. The correctness of this protocol follows from the correctness of the DPRF scheme, with the only additional argument being that we rely on the actual structure of the CL framework where we encrypt the input under f and then finally use CLSolve to recover the actual sum.

Theorem 5. *Construction 3 is Server-Indistinguishable under Chosen Plaintext Attack (S-IND-CPA) provided* DPRF *is a secure key-homomorphic distributed PRF with advantage* ϵ_{DPRF} *then:*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{OPA}} \leqslant 2 \cdot n \cdot \epsilon_{\mathsf{DPRF}}$$

Proof. Let S_j be the event that the adversary outputs 1 in Hybrid_j for each $j \in \{0, \ldots, 2\}$. The proof proceeds through a sequence of hybrids. We define the hybrids below:

$\mathsf{Hybrid}_0(\kappa)$**:** This corresponds to the game where $b = 0$, i.e., for all the honest users in $\mathcal{H}$, the first input is encrypted.

$\mathsf{Hybrid}_1(\kappa)$**:** In this game, we will first guess an index i^*. Implicitly, this is the i^* for which we will inject the random challenge from the DPRF security game. If there is a corruption query issued on i^*, the game aborts. Otherwise, the game responds to $\mathcal{O}_{\mathsf{Enc}}, \mathcal{O}_{\mathsf{Com}}$ with some changes.

- If there is a $\mathcal{O}_{\mathsf{Enc}}(i^*, x_{i^*,\ell}, \ell, \mathcal{S})$, rather than setting $h_{i,\ell}$ to be the evaluation to DPRF, it will instead be a random value in $Y_{i^*,\ell} \leftarrow\!\!\$\ \mathbb{G}$. The rest of it proceeds as before with the adversary receiving aux corresponding to *only* elements in $\mathcal{S}$ and the ciphertext is now set to $\mathsf{ct}_{i^*,\ell} = f^{x_{i^*,\ell}} \cdot Y_{i^*,\ell}$. The inputs to $\mathcal{O}_{\mathsf{Enc}}$ along with the outputs are recorded.

- If there is a call to $\mathcal{O}_{\mathsf{Com}}$ with input label ℓ, then we look at the table of encryption queries. If there exists an entry for i^* with ℓ, then the combination procedure is modified. We will set $\mathsf{AUX}_\ell = (\prod f^{\sum x})^{-1} \cdot \prod \mathsf{ct}$ where x and ct are retrieved from the encryption table.
- Challenge Phase: If i^* is in $\mathcal{H}$, then the encryption procedure for challenge ciphertext for i^* is again modified to use a random $Y_{i^*,\tau} \leftarrow\!\!\$\ \mathbb{G}$. We again need to modify AUX_τ to be consistent in order to ensure that the adversary can perform $\mathsf{Aggregate}$ on its own.

Claim: If DPRF *is a secure, distributed PRF with advantage* ϵ_{PRF}*, then,*

$$|\Pr[S_0] - \Pr[S_1]| \leqslant n \cdot \epsilon_{\mathsf{DPRF}}$$

The proof of this claim is fairly straightforward. Note that our proof of security of the DPRF hinged on a single challenge. It is fairly straight forward to reduce the multi-challenge version of DPRF security game to the single-challenge version presented in Definition 12. This comes at the price of q_c where q_c is the total number of queries made to the challenge oracle. Let $\mathcal{A}$ be capable of distinguishing Hybrid_0 and Hybrid_1, then we can build $\mathcal{B}$ that can win in the DPRF security game. $\mathcal{B}$ does the following:

- Does not corrupt any shares.
- Every encryption query with i^* for which $\mathcal{A}$ expects $\left\{\mathsf{aux}^{(j)}_{i^*,\ell}\right\}_{j\in\mathcal{S}}$, $\mathcal{B}$ issues Eval queries for $j \in \mathcal{S}$ at ℓ as input. Meanwhile, to mask inputs corresponding to i^*, $\mathcal{B}$ simply invokes challenge oracle with the ℓ.
- For the challenge phase, again, rather than sampling the value, it uses its challenge oracle to get the masking value.

Note that if $\mathcal{B}$'s challenger's tossed bit was 0, then $\mathcal{B}$ perfectly simulates Hybrid_0 while simulating Hybrid_1 should the tossed bit be 1. This concludes the proof.

$\mathsf{Hybrid}_2(\kappa)$: Same as Hybrid_1, except that the value encrypted corresponds to bit $b = 1$ for all the honest parties' challenge inputs. *Claim:* Hybrid_1 *and* Hybrid_2 *are identically distributed.*

$\mathsf{Hybrid}_3(\kappa)$: Same as Hybrid_2, except we now go back to honest generation of ciphertexts for i^*. *Claim: If* DPRF *is a secure, distributed PRF with advantage* ϵ_{PRF}*, then,*

$$|\Pr[S_2] - \Pr[S_3]| \leqslant n \cdot \epsilon_{\mathsf{DPRF}}$$

The proof of this claim is akin to the earlier claim.

□

We defer our exposition on OPA constructions from $\mathtt{LWR}$ to Section D.2. We also present a "robust" version whereby the server can detect malicious behavior and remove those clients. This is discussed in Section F.

8 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 $\mathtt{t}$ of them), then the client's input remains hidden. It is easy to observe that our construction of OPA 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, we need to modify our existing OPA syntax and construction. 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. The oracle definitions are changed here. $\mathcal{O}_{\mathsf{Corr}}$ proceeds as before 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 $[n]\backslash\mathbf{K}$. 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 7 (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:*

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

8.1 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 $\mathsf{C\text{-}Combine}$ 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, $\mathsf{S\text{-}Combine}$ 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.

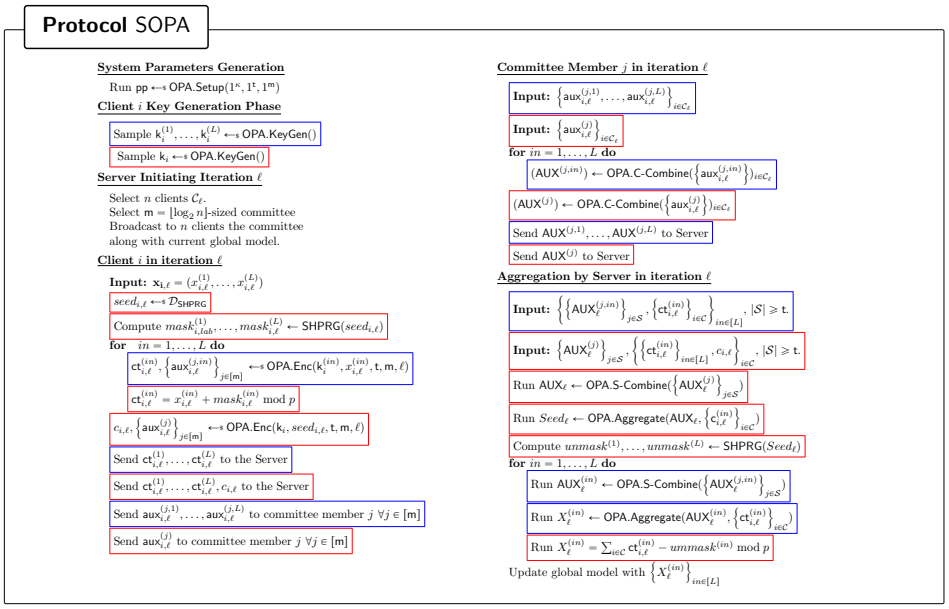

Figure 4: In this figure, we present our two constructions of SOPA. The text inside of the blue box corresponds to SOPA_1 while SOPA_2 is contained in the red box.

9 Secure Aggregation for Federated Learning with Single Client Interaction

As discussed, the motivating application for OPA is privacy-preserving federated learning (PPFL), allowing the server to aggregate models across iterations with dynamic client participation. For ease of presentation, we have defined OPA to work over single client input. However, for PPFL, the client usually has to work with multiple inputs. In this section, we introduce SOPA, the streaming version of OPA. The clients' masked inputs remain private, with the server only learning the sum. Our goal is a protocol without distributed setup, which is required before each iteration in prior works like [18, 12, 74, 57, 72, 70]. Some allow setup reuse but OPA eliminates this entirely.

In our protocol, we will set the committee of size to be $\mathsf{m} = \lfloor \log_2(n) \rfloor$. Our protocol supports arbitrary threshold of reconstruction, but our implementation will set choice to be $\mathsf{t} = 2 \cdot m/3 + 1$. We present two approaches to mask vectors of length L. In Section 9.1, we naively mask each element and in Section 9.2 we discuss how to improve communication complexity using seed homomorphic PRG. Meanwhile, in Section H, we discuss the asynchronous setting where messages to intended recipients may be delayed or dropped. This requires an additional round of interaction between server and committee members to agree on a set of online clients whose inputs can be aggregated with the help of committee.

9.1 Non-Interactive, Single Round, Secure Aggregation Protocol

In the synchronous setting, the chosen committee members and server receive respective information from all online clients per round. Intuitively, this corresponds with each client running OPA protocol L times, once per input element. The committee members combine all client information received for that round (without additional randomness) and communicate a succinct version to the server which helps server aggregate. Our protocol, which we call SOPA_1, is built from any secure OPA protocol. The details are provided in Figure 4. Those lines that are specifically executed by SOPA_1 is shown with a blue box around the line.

Remark 5. (Packed Secret Sharing) This can be optimized using Packed Secret Sharing. OPA.Enc does: (a) encrypts information and (b) secret-shares the key. In SOPA_1, each client has L keys,

shared naively. Instead, Packed Secret Sharing (Construction 5) over integers has the client send just one $\mathsf{aux}_{i,\ell}^{(j)}$ to each committee member j. Similarly, the committee avoids the loop and can simply combine one element, per client. The server still reconstructs the L aggregated keys. In Tables 2 and 3, this version is represented as $\mathsf{SOPA}_1^{(p)}$.

9.2 Non-Interactive, Single Round, Secure Aggregation Protocol Using Seed Homomorphic PRG

In this section, we rely on a dual-pronged strategy first proposed in SASH [72]. First, a client samples a single key k_i by running $\mathsf{PSA.KeyGen}$. Then, it samples a seed $seed_i$ from the domain of a seed homomorphic PRG. The formal definition is presented in Section E.1 but informally the property is that $\mathsf{SHPRG}(seed) + \mathsf{SHPRG}(seed') = \mathsf{SHPRG}(seed + seed')$. To encrypt a vector of length L, client i expands the seed $seed_i$ to a masking vector $\mathbf{mask}_i$ of length L. Inputs are masked by simply doing $\mathbf{mask_i} + \mathbf{x}_i$. Meanwhile, the $seed_i$ itself is encrypted using $\mathsf{PSA.Enc}$ By correctness of OPA protocol, we have that the server can efficiently recover $\sum_{i=1}^{n} seed_i$. Note that, by correctness of a seed homomorphic PRG has the property that: $\mathsf{SHPRG}(\sum_{i=1}^{n} seed_i) = \sum_{i=1}^{n} \mathsf{SHPRG}(seed_i) = \sum_{i=1}^{n} \mathbf{mask}_i$. Therefore, the server can expand the aggregated seed to compute the vector sum of all of the masks, which can be used to unmask the ciphertexts. The immediate benefit of this version of the protocol is that the masked inputs are no longer group elements, but merely integers/field elements which improves the computation and communication. Unfortunately, they had to rely on an approximate, i.e., almost seed homomorphic PRG [20] based on the $\mathtt{LWR}$ assumption. Formally, it is presented as Construction 10. Informally, $\mathsf{SHPRG}(s) := \left\lceil \mathbf{A}^T \cdot s \right\rfloor_p$ where n, m, p, q satisfy $p < q, n < m$ are public parameters such that $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n \times m}$. This protocol is only almost seed homomorphic, i.e., $\mathsf{SHPRG}(\sum_{i=1}^{n} seed_i) = \sum_{i=1}^{n} \mathsf{SHPRG}(seed_i) + e, e \in \{0, \ldots, n-1\}^L$ Due to the presence of the error value, we need to ensure that the choice of p is sufficiently large that one can round down efficiently. Specifically, one requires that $p > \sum_{i=1}^{n} x_{i,\ell} + e$. This leads to an immediate trade-off where $O(L)$ work done by the client and server would be over field elements rather than group elements, vastly improving the efficiency. Unfortunately, careful consideration has to be made for the choice of parameters. For completeness, we also present an SHPRG construction that is secure under the $\mathtt{HSM}_{\mathsf{M}}$ assumption next in Section E.

We formally present our construction of SOPA_2 in Figure 4 that is built from any OPA and any SHPRG. Those lines that are solely executed by SOPA_2 are encased in a red box.

10 Experiments

In this section, we perform different experiments to demonstrate that $(\mathsf{t}, \mathsf{m}, \mathsf{M} = p)$ OPA, (hereafter $\mathsf{OPA}_{\mathsf{CL}}$) based on the CL framework, can indeed be used as a secure aggregation algorithm and to train machine learning models. 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 [27] 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 to the "global" setting in ABIDES's default parameters (the range is 21 microseconds to 53 milliseconds), and use the default parameters for the jitter. This framework was used to measure performance of other prior work including [74, 57].

Microbenchmarking Secure Aggregation. Our first series of experiments is to run $\mathsf{OPA}_{\mathsf{CL}}$ to build a secure aggregation protocol. We also compare with existing work including [18, 12, 57, 74]. We vary the offline rates, 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

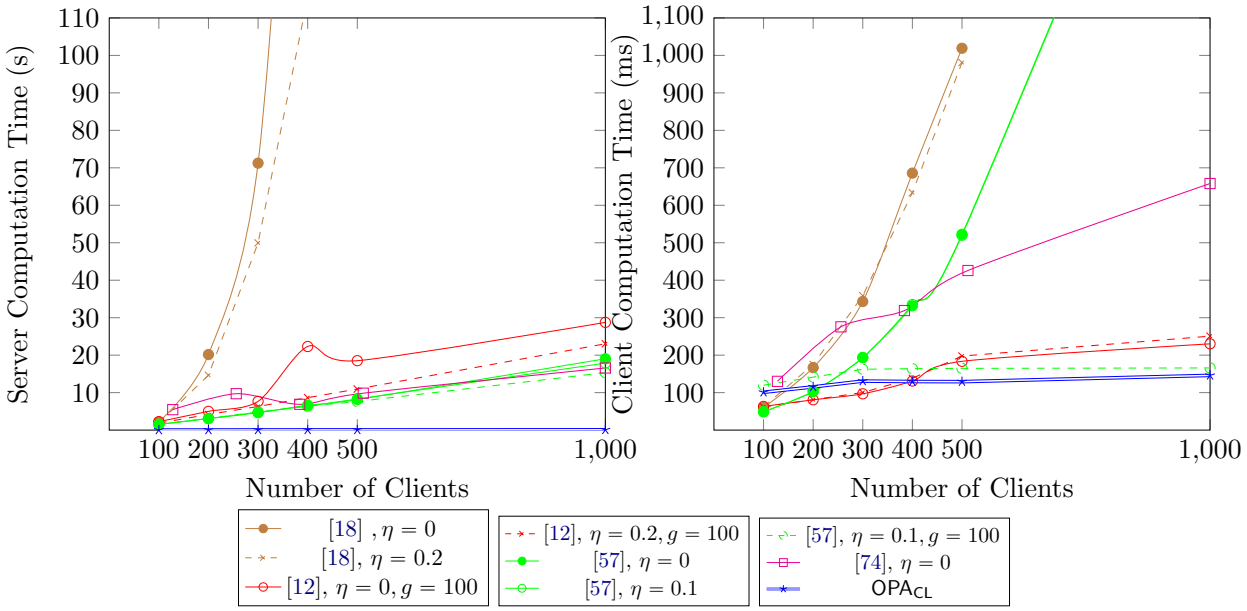

Figure 5: Client and Server Computation Time as a function of client count across different algorithms.

size of the neighborgood or group. For [57], we set the input size to be bounded by 10^4. Recall that [57] does not have efficient aggregate recovery and requires input bounding. Also, [57] 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 5, $\mathsf{OPA}_{\mathsf{CL}}$'s server running time is less than 1 second - owing to a single round protocol with support of efficient recovery of the aggregate. Our client performance scales best for a large number of clients while competitive for fewer. This beats all other protocols. Each committee member computes $< 1ms$. $\mathsf{OPA}_{\mathsf{CL}}$ assumes concurrent client communication to committee and server. Additional checks like in Section H where server and committee agree on client intersection to compute aggregation are possible. These would have (a) no impact on client computation time, and (b) a negligible increase in server/committee computation time. [72] combines existing secure aggregation protocol [18] with a seed-homomorphic PRG to gain efficiency for large input sizes. However, their cost is dominated by [18], which we significantly outperform. One could combine SASH with $\mathsf{OPA}_{\mathsf{CL}}$ to achieve efficient round-communication and improved server computation, optimizing input size scaling. With our 256-bit prime to initialize the CL group, each group element requires 56 bytes. One can further reduce bandwidth by using the "New Key" mode and sending shares of keys instead. Note that SOPA is instantiated from any OPA protocol, including the LWR based construction. However, while [72] showed how to use LWR based SHPRG to build a secure aggregation protocol, demonstrating accuracy in training datasets, care must be taken to ensure that the error growth can be controlled. Our experiments will rely on the $\mathtt{HSM}_{\mathsf{M}}$ based construction and we leave it as future work to instantiate a protocol based on $\mathsf{OPA}_{\mathtt{LWR}}$.

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 $\mathsf{CAPS}_{\mathsf{CL}}$ 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) [75] which

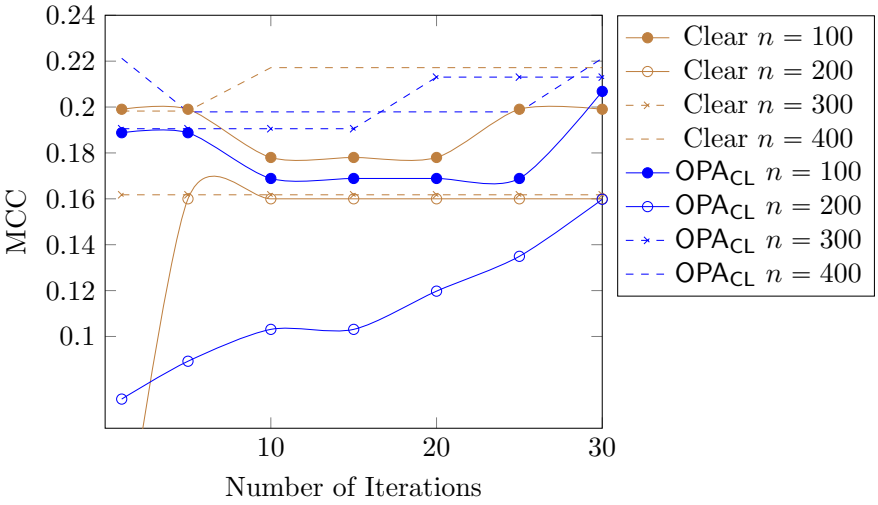

Figure 6: Measure of MCC as a function of number of iterations and number of clients, comparing $\mathsf{OPA}_{\mathsf{CL}}$ and plaintext learning.

better evaluates binary classification with unbalanced classes.

- Adult Census Dataset: We first run experiments on the adult census income dataset from [28, 62] 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 [81], 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 6 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.

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, Heidelberg, 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, Heidelberg, May 2012.

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

[6] 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, Heidelberg, May 2023.

[7] 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. https://eprint.iacr.org/2022/181.

[8] 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, Heidelberg, April 2012.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 227–247. Springer, Heidelberg, December 2019.

[15] Jean-François Biasse, Michael J. Jacobson, and Alan K. Silvester. Security estimates for quadratic field based cryptosystems. In Ron Steinfeld and Philip Hawkes, editors, *ACISP 10: 15th Australasian Conference on Information Security and Privacy*, volume 6168 of *Lecture Notes in Computer Science*, pages 233–247. Springer, Heidelberg, July 2010.

[16] Jean-François Biasse. Improvements in the computation of ideal class groups of imaginary quadratic number fields, 2010.

[17] 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, Heidelberg, August 2023.

[18] 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.

[19] 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, Heidelberg, August 2019.

[20] 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, Heidelberg, August 2013.

[21] 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.

[22] 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, Heidelberg, August 2023.

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

[24] Johannes Buchmann and Ulrich Vollmer. *Binary quadratic forms - an algorithmic approach*, volume 20 of *Algorithms and computation in mathematics*. Springer, 2007.

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

[26] 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, Heidelberg, May 2020.

[27] 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.

[28] 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.

[29] Jan Camenisch and Victor Shoup. Practical verifiable encryption and decryption of discrete logarithms. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer, Heidelberg, August 2003.

[30] 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, Heidelberg, August 2019.

[31] 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, Heidelberg, May 2020.

[32] 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, Heidelberg, August 2017.

[33] 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, Heidelberg, April 2015.

[34] 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, Heidelberg, December 2018.

[35] Guilhem Castagnos, Fabien Laguillaumie, and Ida Tucker. Threshold linearly homomorphic encryption on $\mathbf{Z}/2^k\mathbf{Z}$. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022, Part II*, volume 13792 of *Lecture Notes in Computer Science*, pages 99–129. Springer, Heidelberg, December 2022.

[36] 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, Heidelberg, April / May 2018.

[37] Jihoon Cho, Jincheol Ha, Seongkwang Kim, ByeongHak Lee, Joohee Lee, Jooyoung Lee, Dukjae Moon, and Hyojin Yoon. Transciphering framework for approximate homomorphic encryption. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part III*, volume 13092 of *Lecture Notes in Computer Science*, pages 640–669. Springer, Heidelberg, December 2021.

[38] 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.

[39] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Dynamic decentralized functional encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 747–775. Springer, Heidelberg, August 2020.

[40] 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, Heidelberg.

[41] 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, Heidelberg, August 2000.

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

[43] 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.

[44] 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.

[45] 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, Heidelberg, October 2021.

[46] D.A. Cox. *Primes of the Form x2+ny2: Fermat, Class Field Theory, and Complex Multiplication*. Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley, 2014.

[47] 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, Heidelberg, April 2006.

[48] Parthasarathi Das, Michael J Jacobson, and Renate Scheidler. Improved efficiency of a linearly homomorphic cryptosystem. In *Codes, Cryptology and Information Security: Third International Conference, C2SI 2019, Rabat, Morocco, April 22–24, 2019, Proceedings-In Honor of Said El Hajji 3*, pages 349–368. Springer, 2019.

[49] 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, Heidelberg, December 2021.

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

[51] T. Elgamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[52] Keita Emura. Privacy-preserving aggregation of time-series data with public verifiability from simple assumptions. In Josef Pieprzyk and Suriadi Suriadi, editors, *ACISP 17: 22nd Australasian Conference on Information Security and Privacy, Part II*, volume 10343 of *Lecture Notes in Computer Science*, pages 193–213. Springer, Heidelberg, July 2017.

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

[54] Frank A. Feldman. Fast spectral tests for measuring nonrandomness and the DES. In Carl Pomerance, editor, *Advances in Cryptology – CRYPTO'87*, volume 293 of *Lecture Notes in Computer Science*, pages 243–254. Springer, Heidelberg, August 1988.

[55] 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.

[56] 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. https://eprint.iacr.org/2022/942.

[57] 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. https://eprint.iacr.org/2022/714.

[58] James L Hafner and Kevin S McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American mathematical society*, 2(4):837–850, 1989.

[59] 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.

[60] 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.

[61] Michael J. Jacobson. Computing discrete logarithms in quadratic orders. *J. Cryptol.*, 13(4):473–492, jan 2000.

[62] 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.

[63] Marc Joye and Benoît 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, Heidelberg, April 2013.

[64] 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, Zaïd 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.

[65] 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, Paper 2023/451, 2023. https://eprint.iacr.org/2023/451.

[66] Thorsten Kleinjung. Quadratic sieving. *Math. Comput.*, 85(300):1861–1873, 2016.

[67] 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, Heidelberg, August 2019.

[68] 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, Heidelberg, October 2014.

[69] Iraklis Leontiadis, Kaoutar Elkhiyaoui, Melek Önen, and Refik Molva. PUDA - privacy and unforgeability for data aggregation. In Michael Reiter and David Naccache, editors, *CANS 15: 14th International Conference on Cryptology and Network Security*, Lecture Notes in Computer Science, pages 3–18. Springer, Heidelberg, December 2015.

[70] 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*, pages 302–334, Singapore, 2023. Springer Nature Singapore.

[71] 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, Heidelberg, June 2012.

[72] 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.

[73] 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.

[74] 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.

[75] 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.

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

[77] 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, Heidelberg, May 1999.

[78] Dinh Duy Nguyen, Duong Hieu Phan, and David Pointcheval. Verifiable decentralized multi-client functional encryption for inner product. To Appear in Asiacrypt 2023, 2023.

[79] Pascal Paillier. Trapdooring discrete logarithms on elliptic curves over rings. In Tatsuaki Okamoto, editor, *Advances in Cryptology – ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 573–584. Springer, Heidelberg, December 2000.

[80] 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.

[81] 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.

[82] 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, Heidelberg, November 2022.

[83] 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.

[84] 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.

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

[86] István András Seres, Péter Burcsi, and Péter Kutas. How (not) to hash into class groups of imaginary quadratic fields? Cryptology ePrint Archive, Paper 2024/034, 2024. https://eprint.iacr.org/2024/034.

[87] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, nov 1979.

[88] 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.

[89] 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, Heidelberg, May 2000.

[90] 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.

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

[92] 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.

[93] 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.

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

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

[96] 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, Heidelberg, May 2019.

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

[98] 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, Heidelberg, May 2021.

[99] Yulin Zhao, Hualin Zhou, and Zhiguo Wan. Superfl: Privacy-preserving federated learning with efficiency and robustness. Cryptology ePrint Archive, Paper 2024/081, 2024. https://eprint.iacr.org/2024/081.

A Class Groups and Cryptography

In this section, we present an abridged overview of class groups of imaginary quadratic fields. We refer the readers to the work of Buchmann and Vollmer [24] and Cox [46] for a comprehensive treatment.

A.1 Class Groups

Quadratic Fields. Let $\mathbb{Q}$ be the set of rational numbers. Then, imaginary quadratic fields are extensions of degree 2 over $\mathbb{Q}$ represented as $K = \mathbb{Q}(\sqrt{D})$ where $D < 0$ and is a square-free integer. With every such field, one can identify an integer called *fundamental discriminant* and is denoted by K with the following definition: $K = \begin{cases} D & \textbf{if } D \equiv 1 (\mathrm{mod} 4) \\ 4D & \textbf{otherwise} \end{cases}$

Orders. The ring $\mathscr{O}_{\Delta_K}$ of algebraic integers in K is also called the *maximal order* and is denoted by $\mathbb{Z}[\omega_K]$ where $\omega_K = (K + \sqrt{K})/2$. We define *orders* as the special subrings of $\mathscr{O}_{\Delta_K}$ and is associated with a nonfundamental discriminant $\Delta_\ell := \ell^2 \Delta_K$ where ℓ is called *conductor*. The order $\mathcal{O}_{\Delta_\ell}$ of the conductor ℓ is the ring $\mathbb{Z}[\ell\omega_K]$

Class Groups. Any discriminant Δ has an associated finite abelian group called *ideal class group*, denoted by $\mathrm{Cl}(\Delta)$, and is defined as the quotient of the group of (invertible fractional) ideals of $\mathscr{O}_\Delta$ by the subgroup of principal ideals. The order of this group is the *class number* and is denoted by $h(\Delta)$.

The value of $h(\Delta)$ is close to $\sqrt{|\Delta|}$ in general. Further, one can evaluate its number of bits in polynomial time using the analytic class formula number as proposed by McCurley [76]. Unfortunately, the only known method to compute $h(\Delta)$ from Δ takes a subexponential time in the length of Δ. This hardness forms the basis of the various cryptographic protocols that have been implemented based on groups of unknown order. The benefit of these protocols is that one can simply generate the integer Δ to use these groups, unlike the group of invertible elements of $\mathbb{Z}/N\mathbb{Z}$ which requires a trusted setup to generate the RSA modulus N as the factorization needs to be kept secret to generate the unknown order group. The other benefit is that discrete logarithms in $\mathrm{Cl}(\Delta)$ are also hard to compute with only known sub-exponential time algorithms [15].

A.2 Class Group and Cryptography

Class Group-based Cryptography originated in the late 1980s with the pioneering work of Buchmann and Williams [25] and McCurley [76]. They proposed that class groups of maximal orders of imaginary quadratic fields could offer better security than multiplicative groups of finite fields, based on the exponential running time of discrete logarithm algorithms for class groups versus subexponential time for finite field groups using index calculus. Subsequent progress on computing class group structure was made by Hafner and McCurley [58], Jacobson [61], Biasse [16], and Kleinjung [66], but costs still increase with the discriminant size, with 512-bit discriminants the current limit [14]. Notably, subexponential class group computation is asymptotically slower than integer factorization ($L_{1/2}(|\Delta|)$ vs $L_{1/3}(N)$ for factoring integer N). Biasse et al.[15] conjectured 1872-bit discriminants are needed for 128-bit security, versus 3072-bit RSA moduli.

Despite early vulnerabilities, class group cryptography has seen a resurgence in diverse applications over the past decade. A major advance was Castagnos and Laguillaumie's cryptosystem using a class group subgroup with easy discrete logs [33]. This became the foundation for various protocols like projective hash functions for inner product encryption [34], threshold ECDSA signatures [30, 31], coin mixing [56], and timed commitments [93]. It also enabled multiparty computation from threshold encryption [22] and verifiable secret sharing [65]. A key feature of class groups is suitability for multiparty protocols with one-time transparent setup, no interaction. This

Table 5: Comparison of the RSA Modulus Size and the Class Group Size for various Security Levels. All the sizes are in bits. The RSA Modulus forms the basis of the DCR-based construction.

Security Level	RSA Size Modulus	Class Group Size
112	2048	1348
128	3072	1872
192	7680	3598
256	15360	5971

has allowed verifiable random functions without trusted setup [96, 97], accumulators [71, 19], encryption switching [32], designated verifier NIZK [36], SNARKs [67, 26, 6], homomorphic secret sharing, correlation functions [1], range proofs [45, 44], and vector commitments [7].

B Secret Sharing

Definition 8 (Secret Sharing over $\mathbb{Z}$)**.** *A* (t, 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*, and the threshold* t *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 by which is denoted by* $\Delta = \mathsf{m}!$*, and the following syntax:*

- $(\mathsf{s}_1, \ldots, \mathsf{s}_\mathsf{m}) \leftarrow\!\!\$\ \mathsf{Share}(\mathsf{s}, \mathsf{m}, \mathsf{t})$*: 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* t *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* t *shares, the reconstruction algorithm outputs the secret* s'*.*

We further require the following security properties.

- *Correctness: For any* $\mathsf{m}, \kappa_s, \mathsf{t}, \ell_s, \ell_r \in \mathbb{Z}$ *with* $\mathsf{t} < \mathsf{m}$*, and any set* $\mathcal{S} \subseteq [m]$ *with* $|\mathcal{S}| \geqslant \mathsf{t}$*, for any* $\mathsf{s} \in \mathbb{Z}$ *such that* $\mathsf{s} \in [0, 2^{\ell_s})$ *the following holds:*
$$\Pr\left[\mathsf{s} = \mathsf{s}' \;\middle|\; \begin{array}{c}(\mathsf{s}_1, \ldots, \mathsf{s}_m) \leftarrow\!\!\$\ \mathsf{Share}(\mathsf{s}, \mathsf{m}, \mathsf{t}) \\ \mathsf{s}' \leftarrow \mathsf{Reconstruct}(\{\mathsf{s}_i\}_{i \in \mathcal{S}})\end{array}\right]$$
- *Statistical Privacy [47]: We say that a* (t, 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*.*

B.1 Secret Sharing over Integer Space

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 [87] 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 $g^s = g^{\sum_{i\in\mathcal{S}} \alpha_i \cdot \mathsf{s}_i} = \prod_{i\in\mathcal{S}} g_i^{\alpha_i}$. Unfortunately, the same protocol does 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 present 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.* [22].

Construction 4 (Shamir's Secret Sharing over $\mathbb{Z}$)**.** Consider the following (t, m) Integer Secret Sharing scheme where m is the number of parties and t 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:

$\mathsf{Share}(\mathsf{s}, \mathsf{t}, \mathsf{m})$

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

$(r_1, \ldots, r_{\mathsf{t}-1}) \leftarrow_{\$} [0, 2^{\ell_r + \kappa_s})$

$f(X) := \tilde{\mathsf{s}} + \sum_{i=1}^{\mathsf{t}-1} r_i \cdot X^i$

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

$\mathsf{GetCoeff}(\mathcal{S})$

if $|\mathcal{S}| \geqslant \mathsf{t}$

  for $i \in \mathcal{S}$ **do**

    $\Lambda_i := \prod_{j\in\mathcal{S}\setminus\{i\}} \frac{x_j}{x_j - x_i} \cdot \Delta$

  return $\{\Lambda_i\}_{i\in\mathcal{S}}$

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

if $|\mathcal{S}| \geqslant \mathsf{t}$

  $\{\Lambda_i\}_{i\in\mathcal{S}} \leftarrow \mathsf{GetCoeff}(\mathcal{S})$

  $\mathsf{s}' := \sum_{i\in\mathcal{S}} \Lambda_i \cdot \mathsf{s}^{(i)}$

  return s'

We omit the proof of correctness as it is similar to the original Shamir's Secret Sharing scheme with the only difference being that the shared secret is now $\mathsf{s} \cdot \Delta$ and the Lagrange coefficients can only reconstruct to this value. 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 6 ([22])**.** *Construction 4 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 [22, §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 5 (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 $\left\{\mathsf{s}^{(i)}\right\}_{i \in [\mathsf{m}]}$

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

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 9 (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 7 (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 9.*

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 8. *Construction 5 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 9 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 $\ell_r \geqslant \ell_s + \lceil \log_2(\mathsf{sp}_{\max} \cdot (t-1) \cdot \rho) \rceil + 1$ □

C Pseudorandom Functions

Definition 10 (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 11 ((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:* $\mathsf{Eval}(\mathsf{k}_1, x) \times \mathsf{Eval}(\mathsf{k}_2, x) = \mathsf{Eval}(\mathsf{k}_1 \oplus \mathsf{k}_2, x) + e$*.*

Definition 12 (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.*

- $\mathsf{k}^{(1)}, \ldots, \mathsf{k}^{(\mathsf{m})} \leftarrow_{\$} \mathsf{Share}(\mathsf{k}, \mathsf{t}, \mathsf{m})$*: On input of the number of parties* m*, threshold* t*, 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* $\mathsf{m}, \mathsf{t}, \kappa \in \mathbb{Z}$ *with* $\mathsf{t} \leqslant \mathsf{m}$ *and any set* $\mathcal{S} \subseteq [m]$ *with* $|\mathcal{S}| \geqslant \mathsf{t}$*, any input* $x \in \mathcal{X}$*:*

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

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

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

where:

$$\frac{\mathcal{O}_{\mathsf{Eval}}(i, x)}{\mathbf{E} := \mathbf{E} \cup \{(i, x)\}}$$
$$\textbf{return } \mathsf{P\text{-}Eval}(\mathsf{k}^{(i)}, x)$$

- *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* $|\mathcal{S}| \geqslant \mathsf{t}$*,* $\mathsf{Combine}\left(\left\{y_{1,2}^{(j)}\right\}_{j\in\mathcal{S}}\right) = \mathsf{Eval}(\mathsf{k}_1 \oplus \mathsf{k}_2, x)$

D Lattice-Based Cryptography and Constructions

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 [85]. LWR was introduced by Banerjee *et al.* [8].

Definition 13 (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:*

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

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

We also have the construction from Boneh *et al.* [20] of an *almost* Key Homomorphic PRF from LWR in the Random Oracle model which was later formally proved secure by Ernst and Koch [53] with $\gamma = 1$.

Construction 6 (Key Homomorphic PRF from LWR)**.** Let $\mathtt{H} : \mathcal{X} \to \mathbb{Z}_q^\rho$. Then, define the efficiently computable function $F : \mathcal{X} \times \mathbb{Z}_q^\rho \to \mathbb{Z}_p$ as $\lfloor\langle \mathtt{H}(x), \mathbf{k}\rangle\rceil_p$. F is an almost key homomorphic PRF with $\gamma = 1$.

D.1 Distributed PRF from LWR

Let us revisit Construction 6. 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 define now:

Construction 7 (Secret Sharing over $\mathbb{F}_q$)**.**

- $\mathsf{SecretShare}(\mathsf{k}, \mathsf{t}, \mathsf{m})$: Sample a random polynomial $f(X) \in \mathbb{F}_q[X]$ of degree $\mathsf{t}-1$ such that $f(0) = \mathsf{k}$. Then, **return** $\{f(j)\}_{j\in[\mathsf{m}]}$
- $\mathsf{Coeff}(\mathcal{S})$: On input of a set $\mathcal{S} = \{i_1, \ldots, i_{\mathsf{t}}, \ldots\} \subseteq [\mathsf{m}]$ of at least t indices, compute $\lambda_{i_j} = \prod_{\zeta\in[\mathsf{t}]\setminus\{j\}} \frac{i_\zeta}{i_\zeta - i_j}$. Then, **return** $\left\{\lambda_{i_j}\right\}_{i_j\in\{i_1,\ldots,i_{\mathsf{t}}\}}$

Now let us look at some error propagation when applying the almost key homomorphic PRF. We have that for $\mathbf{k}_1, \mathbf{k}_2 \leftarrow\!\!\$\ \mathbb{Z}_q^\rho$,

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

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 $e_T \in 0, \ldots, T$. This becomes a cause for concern as, in the threshold construction using the Shamir Secret Sharing over the field as shown in Construction 7, 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 $e_{\lambda_{i_j}} \in \{0, \ldots, i_j\}$. 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 4 by essentially clearing the denominator by multiplying with $\Delta := \mathsf{m}!$. This is a technique made popular by the work of Shoup [89] 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.* [20]. [5] Then, the combine algorithm will simply multiply all partial evaluations with Δ as well.

Construction 8 (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 7.

Issues with the Construction from Boneh et al. [20, §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 Thoerem 3 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.

[5] 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.

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

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

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

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

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

Figure 7: Construction of Distributed PRF based on the Secret Sharing scheme of Construction 7 where $\mathsf{SS} = (\mathsf{SecretShare}, \mathsf{Coeff})$ with $\mathsf{pp}_{\mathsf{SS}}$ denoting the public parameters of the secret sharing scheme.

Correctness.

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

The last step follows provided the error term is small. Recall that $e_\mathsf{t} \in \{0, \ldots, \mathsf{t}\}$ and $e_{\lambda_{i_j}} \in \{0, \ldots, \lambda_{i_j}\}$. 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,

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

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

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

Pseudorandomness. The proof of pseudorandomness follows the outline of the proof of Theorem 3 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,

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

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 $\mathsf{P\text{-}Eval}$ 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 $\mathtt{LWR}$ Assumption, building it based on the Key Homomorphic, Distributed PRF as presented in Construction 6. 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 $\lfloor u/u' \rfloor > n - 1$.

Protocol One-shot Private Aggregation

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

Run $\mathsf{pp}_{\mathsf{DPRF}} \leftarrow \mathsf{DPRF.Gen}(1^\kappa, 1^\mathsf{t}, 1^\mathsf{m})$
return $\mathsf{pp}_{\mathsf{DPRF}}$

$\underline{\mathsf{KeyGen}()}$

Sample $\mathbf{k} \leftarrow\!\!\$\ \mathcal{K}$
return $\mathbf{k}$

$\underline{\mathsf{Enc}(\mathsf{pp}, \mathbf{k}_i \in \mathbb{Z}_q^\rho, x_i, \mathsf{t}, \mathsf{m}, \ell)}$

Compute $h_{i,\ell} = \mathsf{DPRF.Eval}(\mathbf{k}_i, \ell)$
Compute $\mathsf{ct}_{i,\ell} = (x_i \cdot n + 1) + h_{i,\ell}) \bmod v$
Compute $(\mathbf{k}_i^{(j)})_{j\in[\mathsf{m}]} \leftarrow\!\!\$\ \mathsf{DPRF.Share}(\mathbf{k}_i, \mathsf{t}, \mathsf{m})$
for $j = 1, \ldots, \mathsf{m}$ **do**
  $\mathsf{aux}_{i,\ell}^{(j)} \leftarrow \left\lfloor \langle \mathtt{H}(\ell), \mathbf{k}_i^{(j)} \rangle \right\rfloor_p$
return $\mathsf{ct}_{i,\ell}, \left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{j\in[\mathsf{m}]}$

$\underline{\mathsf{C\text{-}Combine}(\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{i\in\mathcal{C}})}$

Compute $\mathsf{AUX}_\ell^{(j)} = \left\lfloor \Delta \cdot \sum_{i\in\mathcal{C}} \mathsf{aux}_{i,\ell}^{(j)} \right\rfloor_u$
return $\mathsf{AUX}_\ell^{(j)}$

$\underline{\mathsf{S\text{-}Combine}(\left\{\mathsf{AUX}_\ell^{(j)}\right\}_{j\in\mathcal{S}})}$

Run $\mathsf{AUX}_\ell \leftarrow \mathsf{DPRF.Combine}(\left\{\mathsf{AUX}_\ell^{(j)}\right\}_{j\in\mathcal{S}})$
return AUX_ℓ

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

Compute $X_\ell = \sum_{i\in\mathcal{C}} \mathsf{ct}_{i,\ell} - \mathsf{AUX}_\ell \mod v$
Round up $\lceil X_\ell \rceil$ to X'_ℓ, the nearest multiple of n
return $(X'_\ell - n)/n$

Figure 8: Our Construction of OPA based on the LWR Assumption.

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. Specifically, the client's share to the committee will only be the first level of the evaluation, i.e., rounded down to p. C-Combine will then add the shares up, multiply with the offset, and then round down to u. We will show that provided $\lfloor p/u \rfloor > \Delta \cdot n$, the output of C-Combine is consistent with $\mathsf{P\text{-}Eval}(\sum_{i=1}^n \mathbf{k}_i^{(j)}, x)$. 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 S-Combine outputs $\mathsf{Eval}(\sum_{i=1}^n \mathbf{k}_i, x)$.

Construction 9. We present our construction in Figure 8.

Correctness. Earlier, we showed how the output of S-Combine is $\mathsf{Eval}(\sum_{i=1}^n \mathbf{k}_i, x)$. Now, let us look at Aggregate algorithm.

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

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.

Theorem 9. *Construction 9 is Server-Indistinguishable under Chosen Plaintext Attack provided Construction 8 is a secure PRF.*

The proof of the theorem proceeds largely similar to that of Theorem 5 and is omitted.

D.3 Modified Construction of OPA under LWR Assumption

The proposed syntax for OPA allows for a one-time key generation with every encryption requiring a new key share per committee. However, one could also move the key-sharing to the key generation algorithm and simply perform partial evaluations during the encryption. We opt to include the sharing of the key within the encryption algorithm to allow for greater flexibility should either the threshold or the committee size change during a label.

Indeed, one can also leverage the non-interactivity of the client whereby the client's choice of key(s) does not affect the key choices of any other clients, the committee members, or even the server. Therefore, one can move the generation of keys inside the encryption procedure. For every label, the client samples a new key, encrypts information with the new key, shares the key, and then generates auxiliary information using the shares. While this might be intuitively inefficient, we show that such a process actually affords efficiency in both computation and communication, at least in the LWR construction. It immediately follows that, in this setting, for each label, we essentially run a distinct OPA protocol as we defined before.

More specifically, consider the following variation of the scheme:

- For ℓ, Client i samples $\mathbf{k}_{i,\ell} \leftarrow\!\!\$\ \mathbb{Z}_q^\rho$.
- Compute Packed Secret Sharing over $\mathbb{F}_q$ space to $\mathbf{k}_{i,\ell}$ to get shares $\mathsf{aux}_{i,\ell}^{(j)}$ for $j = 1, \ldots, \mathsf{m}$.
- Masking happens as before by performing: $\mathsf{ct}_{i,\ell} = n \cdot x_{i,\ell} + 1 + \lfloor \langle \mathtt{H}(\ell), \mathbf{k}_i \rangle \rceil_p \bmod p$
- The committee member j receives: $\left\{ \mathsf{k}_{i,\ell}^{(j)} \right\}_{i \in \mathcal{C}}$. It simply adds up all the shares as $\mathsf{AUX}_\ell^{(j)} = \sum_{i \in \mathcal{C}} \mathsf{aux}_{i,\ell}^{(j)} \bmod q$
- The server, upon receiving $\mathsf{AUX}_\ell^{(j)}$ for $j \in \mathcal{S}$ with $|\mathcal{S}| \geqslant \mathsf{t}$ does the following:
  - Reconstruct from the shares $\mathsf{AUX}_\ell^{(j)}$, the values $\mathbf{K}_\ell = (\mathsf{k}_\ell^{(1)}, \ldots, \mathsf{k}_\ell^{(\rho)})$, over the field $\mathbb{F}_q$. Now, note that this, by the correctness of Packed Shamir Secret Sharing over Integers and their additive homomorphism implies that $\mathsf{k}_\ell^{(1)}$ is the sum of all the keys in element 1 across $\mathbf{k}_{i,\ell}, \ldots, \mathbf{k}_{n,\ell}$ where n is the number of clients. In other words, $\mathbf{K}_\ell = \sum_{i=1}^n \mathbf{k}_{i,\ell}$
  - Compute $\mathsf{AUX}_\ell = \lfloor \langle \mathtt{H}(\ell), \mathbf{K}_\ell \rangle \rceil_p$
  - Compute

$$\sum_{i=1}^{n} (\mathsf{ct}_{i,\ell}) - \mathsf{AUX}_\ell \bmod p = \sum_{i=1}^{n} \left(n * (x_{i,\ell}) + 1 + \lfloor \langle \mathtt{H}(\ell), \mathbf{k}_i \rangle \rceil_p \right) - \left\lfloor \langle \mathtt{H}(\ell), \sum_{i=1}^{n} \mathbf{k}_{i,\ell} \rangle \right\rceil_p \bmod p$$

$$= n \cdot \sum_{i=1}^{n} x_{i,\ell} + n + \sum_{i=1}^{n} \lfloor \langle \mathtt{H}(\ell), \mathbf{k}_i \rangle \rceil_p - \left\lfloor \langle \mathtt{H}(\ell), \sum_{i=1}^{n} \mathbf{k}_{i,\ell} \rangle \right\rceil_p \bmod p$$

  - Now, we know that $\sum_{i=1}^n \lfloor \langle \mathtt{H}(\ell), \mathbf{k}_i \rangle \rceil_p = \lfloor \langle \mathtt{H}(\ell), \sum_{i=1}^n \mathbf{k}_{i,\ell} \rangle \rceil_p - e$ where $e \in \{0, \ldots, n-1\}$
  - Or,

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

  provided $\sum_{i=1}^n (\mathsf{ct}_{i,\ell}) < (p-n)/n$

– Then, we round up to the nearest multiple of n and do the recovery as before.

Now, observe that this modified construction does not require any additional rounding down and simply requires that $\sum x_i > (p-n)/n$. However, this does require sampling a new key for every label but can enjoy the benefits of packed secret sharing to send minimal communication to and from the committee.

Therefore, by generating a new key at every round, we have simply employed the original Almost Key Homomorphic PRF construction, rather than the threshold one. Consequently, we avoid partial evaluations and instead simply generate new keys repeatedly. This modification can also be applied to the $\mathtt{HSM}_{\mathsf{M}}$ assumption.

E Seed Homomorphic PRG

E.1 Syntax and Security

Definition 14 (Seed Homomorphic PRG (SHPRG))**.** *Consider an efficiently computable function $G : \mathcal{X} \to \mathcal{Y}$ where $(\mathcal{X}, \oplus), (\mathcal{Y}, \otimes)$ are groups. Then $(G, \oplus, \otimes)$ is said to be a secure seed homomorphic pseudorandom generator (SHPRG) if:*

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

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} b \leftarrow\!\!\$\ \{0,1\}, s \leftarrow\!\!\$\ \mathcal{X} \\ Y_0 = G(s), Y_1 \leftarrow\!\!\$\ cY \\ b' \leftarrow\!\!\$\ \mathcal{A}(Y_b) \end{array} \right] \leqslant \frac{1}{2} + \mathrm{negl}(\kappa)$$

- *For every $s_1, s_2 \in \mathcal{X}$, we have that $G(s_1) \otimes G(s_2) = G(s_1 \oplus s_2)$*

E.2 Construction from LWR Assumption

Construction 10 (SHPRG from LWR Assumption)**.** Let $\mathbf{A} \leftarrow\!\!\$\ \mathbb{Z}_q^{n_1 \times n_2}$, then consider the following seed homomorphic PRG $G : \{0,1\}^{n_1} \to \{0,1\}^{n_2}$ where $n_2 > n_1$ is defined as $G(\mathbf{s}) = \lceil \mathbf{A}^\top \cdot \mathbf{s} \rfloor_p$ 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: $G(\mathbf{s}_1 + \mathbf{s}_2) = G(\mathbf{s}_1) + G(\mathbf{s}_2) + e$ where $e \in \{-1, 0, 1\}^{n_2}$

E.3 Construction in CL Framework

Construction 11 (SHPRG in CL Framework)**.** Let $(\mathsf{CLGen}, \mathsf{CLSolve})$ be the class group framework as defined in Definition 1. Then, let $\mathsf{pp}_{\mathsf{CL}} \leftarrow\!\!\$\ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s})$. Consider the following definition of $\mathcal{X} = \mathcal{D}_H$, $\mathcal{Y} = \mathbb{G}$, $G(s) = h^x$.

Theorem 10. *Construction 11 is a secure seed homomorphic PRG.*

Proof. It is easy to verify the homomorphic property of the above construction. For any s_1, s_2 from $\mathcal{D}_H$, the construction is homomorphic in that $h^{s_1} \cdot h^{s_2} = h^{s_1 + s_2}$.

The proof of pseudorandomness can proceed through a sequence of hybrids as follows:

`Game-0` This corresponds to the case when $b = 0$, i.e., the adversary receives $G(s) = h^s$ for $s \leftarrow\!\!\$\ \mathcal{D}_H$.

`Game-1` This corresponds to the case when the adversary receives g^s for $s \leftarrow\!\!\$\ \mathcal{D}$.

Claim: *`Game-0` and `Game-1` are computationally indistinguishable under the $\mathtt{HSM}_{\mathsf{M}}$ assumption.*

`Game-2` This corresponds to $b = 1$ where the adversary receives $Y \leftarrow\!\!\$\ \mathbb{G}$.

Claim: *`Game-1` and `Game-2` are statistically indistinguishable, by definition of $\mathcal{D}$.*

□

F Verifiable One-shot Private Aggregation

We now show how to add verifiability to the aforementioned modified construction of OPA, where a new key is generated per encryption. We focus on the $\mathtt{HSM}_{\mathsf{M}}$ construction, specifically where $\mathsf{M} - p$, for some prime p. We directly present our construction, without the abstraction of the Distributed PRF. Instead, our masking PRF will be based on Construction 1, where $F(k, x) = H(x)^k$. We will also be using the Integer Secret Sharing Scheme from Construction 4. Our construction will use Feldman's Verifiable Secret Sharing scheme [54], adapted to the integer setting, as shown by Braun *et al.* [22, Protocol 2].

This construction aims to detect malicious behavior by either the client or the committee members. To avoid malicious behavior by the clients, they are expected to send commitments to the sharing polynomial and their actual key. For purposes of reporting, one can assume that the client is expected to sign this information. Upon receiving the share of the key, committee members can use the commitments to simply evaluate the polynomial, in the exponent, at point j. If the verification fails, the committee member can simply send the signed message to the server. The server can run the same verification algorithm and then take action against the misbehaving client.

A similar check can be performed by the server to detect errant behavior on the part of the committee members, where the detection would be using the sum-of-shares (contained in $\mathsf{AUX}_\ell^{(j)}$) and the commitments it has already received from the clients. This detects cases where $\mathsf{AUX}_\ell^{(j)}$ is wrongly computed.

Construction 12. We present our construction in Figure 9.

Correctness. Observe that, for a client i, its sharing polynomial is given by:

$$f_i(X) = \Delta \cdot \mathsf{k}_i + \sum_{z=1}^{\mathsf{t}-1} r_i^{(z)} \cdot X^z \Rightarrow \Delta \cdot f_i(X) = \Delta^2 \cdot \mathsf{k}_i + \sum_{z=1}^{\mathsf{t}-1} \Delta r_i^{(z)} \cdot X^z$$

Raising to the exponent by g_F we get:

$$g_F^{\Delta f_i(X)} = \left({g_F}^{\Delta^2 \cdot \mathsf{k}_i}\right) \cdot \prod_{z=1}^{\mathsf{t}-1} \left({g_F}^{\Delta \cdot {r_i}^{(z)}}\right)^{X^z}$$

Plugging in the commitments, as computed, we get:

$$g_F^{\Delta f_i(X)} = \left({C_i^{(0)}}^{\Delta^2}\right) \cdot \prod_{z=1}^{\mathsf{t}-1} \left(C_i^{(z)}\right)^{X^z}$$

Therefore, a share $\mathsf{k}_i^{(j)} = f_i(j)$ should satisfy the equation:

$$g_F^{\Delta \mathsf{k}_i^{(j)}} = \left({C_i^{(0)}}^{\Delta^2}\right) \cdot \prod_{z=1}^{\mathsf{t}-1} \left(C_i^{(z)}\right)^{j^z}$$

Asymptotic Performance. Note that the verification "key" per client, per label consists of t group elements. Therefore, an additional $O(\mathsf{t})$ element is sent from every client to each committee member and to the server. Therefore, the client has an additional $O(\mathsf{m} \cdot \mathsf{t})$ communication complexity. Meanwhile, there is no additional information to be sent by the committee members, but the committee member has to verify, for every client, the correctness of "sharing" which costs a total of $O(n\mathsf{t})$ in computational effort.

Protocol Verifiable, One-shot Private Aggregation

$\underline{\mathsf{Setup}(1^{\kappa_c}, 1^{\kappa_s})}$

Run $\mathsf{pp}_{\mathsf{CL}} \leftarrow \mathsf{CLGen}(1^{\kappa}, 1^{\mathsf{t}}, 1^{\mathsf{m}})$
Sample hash function $\mathtt{H} : \{0,1\}^* \to \mathbb{H}$
Sample a group element $g_F \in \widehat{\mathbb{G}} \backslash \mathbb{F}$
return $\mathsf{pp}_{\mathsf{CL}}, \mathtt{H}, g_F$

$\underline{\mathsf{Enc}(\mathsf{pp}, x_i, \mathsf{t}, \mathsf{m}, \ell)}$

Sample $\mathsf{k}_i \leftarrow\!\!\$\ \mathcal{K}$
Compute $h_{i,\ell} = \mathtt{H}(\ell)^{\Delta^2 \mathsf{k}_i}$
Set $\mathsf{ct}_{i,\ell} = f^{x_i} \cdot h_{i,\ell}$
Compute $\left\{\mathsf{k}_i^{(j)}\right\}_{j\in[\mathsf{m}]} \leftarrow \mathsf{LISS.Share}(\mathsf{k}_i, \mathsf{t}, \mathsf{m})$
Let $r_i^{(1)}, \ldots, r_i^{(\mathsf{t}-1)}$ be the coefficients of the polynomial.
Set $C_i^{(j)} := g_F^{\Delta(r_i{}^{(j)})}$ for $j = 1, \ldots, \mathsf{t}-1$
Set $C_i^{(0)} := g_F^{\mathsf{k}_i}$
$\mathbf{v}_{i,\ell} := (C_i^{(0)}, \ldots, C_i^{(\mathsf{t}-1)})$
return $\mathsf{ct}_{i,\ell}, \mathbf{v}_{i,\ell}$ to Server
return $\mathsf{aux}_{i,\ell}^{(j)} := (\mathsf{k}_i^{(j)}, \mathbf{v}_{i,\ell})$ to Committee Member j

$\underline{\mathsf{C\text{-}Combine}(\left\{\mathsf{aux}_{i,\ell}^{(j)}\right\}_{i\in\mathcal{C}})}$

for $i \in \mathcal{C}$ **do**
  Parse $\mathsf{aux}_{i,\ell}^{(j)} = (\mathsf{k}_i^{(j)}, \mathbf{v}_{i,\ell} = (C_i^{(0)}, \ldots, C_i^{(\mathsf{t}-1)}))$
  if $g_F^{\Delta \cdot \mathsf{k}_i^{(j)}} \neq {C_i^{(0)}}^{\Delta^2} \cdot \prod_{z=1}^{\mathsf{t}-1} (C_i^{(z)})^{(j^z)}$ **then**
    $\mathcal{C} := \mathcal{C} \backslash \{i\}$
    Send to server i's verification failed with proof
Compute $\mathsf{AUX}_\ell^{(j)} = \sum_{i\in\mathcal{C}} \mathsf{k}_i^{(j)}$
return $\mathsf{AUX}_\ell^{(j)}$

$\underline{\mathsf{S\text{-}Combine}(\left\{\mathsf{AUX}_\ell^{(j)}\right\}_{j\in\mathcal{S}}, \{\mathbf{v}_{i,\ell}\}_{i\in\mathcal{C}})}$

for $i \in \mathcal{C}$ **do**
  Parse $\mathbf{v}_{i,\ell} = \left(C_i^{(0)}, \ldots, C_i^{(\mathsf{t}-1)}\right)$
  for $j = 0$ **to** $\mathsf{t}-1$ **do**
    $C^{(j)} := \prod_{i\in\mathcal{C}} C_i^{(j)}$
for $j \in \mathcal{S}$ **do**
  if $g_F{}^{(\Delta \cdot \mathsf{AUX}_\ell{}^{(j)})} \neq {C^{(0)}}^{\Delta^2} \cdot \prod_{z=1}^{\mathsf{t}-1} \left(C^{(j)}\right)^{(j^{(z)})}$ **then**
    $\mathcal{S} = \mathcal{S} \backslash \{j\}$
Run $\mathsf{AUX}_\ell = \mathsf{LISS.Reconstruct}(\left\{\mathsf{AUX}_\ell^{(j)}\right\}_{j\in\mathcal{S}})$
return AUX_ℓ

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

if $g_F^{\mathsf{AUX}_\ell} = {C^{(0)}}^{\Delta^2}$ **then**
  Compute $X_\ell = \prod_{i\in\mathcal{C}} \mathsf{ct}_{i,\ell} \cdot \left(\mathtt{H}(X)^{(\mathsf{AUX}_\ell)}\right)^{(-1)}$
  return $\mathsf{CLSolve}(\mathsf{pp}_{\mathsf{CL}}, X_\ell)$

Figure 9: Our Construction of Verifiable OPA based on the $\mathtt{HSM}_p$ assumption where p is a prime integer that is at least κ_c bits long.

G Private Stream Aggregation and Labeled Decentralized Sum

As discussed before, private stream aggregation [88, 63, 13, 68, 53, 10, 95, 91, 90] has a long line of research which allows the server to compute only the sum of the clients' inputs without leaking information about the actual inputs. An independent line of research has been on leveraging the idea of "labeled decentralized sum" [39, 78] whereby multiple clients encrypt values to a particular label with the final output that is computed being only the sum of inputs encrypted to that label, without any leakage of information about the individual inputs. This tool, dubbed DSUM, finds applications in Decentralized Multi-Client Functional Encryption.

Instantiating PSA with OPA. The versatility of OPA implies that one can simply instantiate the Private Stream Aggregation protocol, in the committee-aided setting. We can call this *Committee Aided Private Stream Aggregation*. The benefit of this is we can now allow clients to join and drop, as needed. It also avoids the pitfalls of a trusted setup for key generation. Finally, some of our constructions (the OPA based on $\mathtt{HSM}_p$ for prime p can also be modified to achieve decentralized parameter generation as shown by the works of Castagnos *et al.* [30, 31].

Labeled Decentralized Sum with OPA. The state-of-the-art DSUM protocol is from the work of Nguyen *et al.* [78]. They take a two-step approach where first they build a one-time decentralized sum protocol from the $\mathtt{HSM}_{\mathsf{M}}$ assumption, and then combine it with Multi-Client Functional Encryption (MCFE) to build DSUM. The former requires a PKI-style infrastructure where the clients' public keys are posted. Meanwhile, OPA helps one achieve the same functionality of private summation, without relying on expensive tools such as MCFE, and by only requiring a committee.

Furthermore, OPA, at its core, can be modified to remove the dependence on the committee while offering DSUM functionality. Quite simply, for each ℓ, there is a new public key for each client i, which is computed as $\mathsf{DPRF.Eval}(\mathsf{k}_i, \ell)$. Then, we can build DSUM using the following construction:

Construction 13 (DSUM from OPA).

- $\mathsf{Setup}(\kappa)$: Run $\mathsf{pp}_{\mathsf{CL}} \leftarrow\!\!\$\ \mathsf{CLGen}(1^{\kappa_c}, 1^{\kappa_s})$
- $\mathsf{KeyGen}(\ell)$: Each client i does the following: (a) sample $\mathsf{k}_i \leftarrow\!\!\$\ \mathcal{K}$, (b) Compute $T_{i,\ell} = \mathsf{DPRF.Eval}(\mathsf{k}_i, \ell)$
- $\mathsf{Encrypt}(x_{i,\ell}, \{T_{i,\ell}\}_{i\in[n]})$: Compute:

$$C_{i,\ell} = f^{x_{i,\ell}} \cdot \left(\prod_{i<j} T_j \prod_{i>j} T_j^{-1}\right)^{\mathsf{k}_i}$$

- $\mathsf{Decrypt}(C_{1,\ell}, \ldots, C_{n,\ell})$: Compute $M = \prod_{i=1}^{n} C_{i,\ell}$ before using $\mathsf{CLSolve}(\mathsf{pp}_{\mathsf{CL}}, M)$ to recover the sum.

Note that the definition of dynamic participation in the context of MCFE is a strictly weaker one than what we employ. The assumption in MCFE is that there is a consensus on the set of public keys per label that is to encrypt for that label.

Robust PSA. Private Stream Aggregation has only operated on the trust model whereby every participant is honest but curious. The only work that deviates from this setting is the work of Emura [52] where they focus on having the aggregator create a proof that is publicly verifiable that the aggregate computed is indeed correct. However, the resulting scheme was based on the DDH construction [88, 13] and thus requires computation of discrete logarithm. Additionally, it requires a pairing-friendly group. Another approach for verifiability was by Leontiadis *et al.* [69] whereby the goal was for the aggregator to prove to a data analyzer that it has computed the correct output.

One can look at another malicious behavior whereby the clients misbehave. The goal is to detect and remove such errant behavior. To this end, one can use our verifiable OPA (defined in Construction 12), as a black box. Thereby, we achieve the first PSA protocol that computes the sum of inputs in a privacy-preserving manner, while removing errant client behavior.

H Two Round Secure Aggregation Protocol for Federated Learning

In the earlier protocol, we were in the synchronous setting which assumed that each committee member receives shares from the same set $\mathcal{C}$ of clients. However, due to network delays, it is possible that some committee members do not receive shares from some of the clients. Unfortunately, this leads to an issue when it comes to reconstruction. To ameliorate, we relax the single round requirement and include some minimal interactivity. Our resulting protocol is still built from our earlier OPA scheme. The key aspect here is that the interaction is only between the committee members and the server. There are no additional rounds or interactivity for the clients. Specifically, rather than combining information about the auxiliary information received by the committee member j, as a first step, the committee member only sends the set of clients whose shares it possesses. Each committee member does the same. Finally, the server computes the intersection of each of the sets sent by the committee member along with the set of clients whose encrypted inputs it has received. The key idea here is that each committee member is expected to respond exactly once per iteration ℓ. This set $\mathcal{C}$ is then sent by the server to the committee members and the server expects a combination with respect to this set $\mathcal{C}$. The rest of the protocol proceeds as before. Our construction, which we dub SOPA_3, is described in Figure 10.

Protocol SOPA_3

One-Time System Parameters Generation

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

Client i Key Generation Phase

Sample $\mathsf{k}_{i,1}, \ldots, \mathsf{k}_{i,L} \leftarrow\!\!\$\ \mathsf{OPA.KeyGen}()$

Data Encryption Phase by Client i in iteration ℓ

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

Set Identification Phase by Committee Member j in iteration ℓ

Let $\left\{\left\{\mathsf{aux}_{i,\ell}^{(j,1)}, \ldots, \mathsf{aux}_{i,\ell}^{(j,L)}\right\}\right\}$ be the inputs received by Committee member j from Clients $i \in \mathcal{C}^{(j)}$
Send $\mathcal{C}^{(j)}$ to server

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 $|\mathcal{S}| \geqslant \mathsf{t}$
Let the server's set of ciphertexts be from clients be denoted by $\mathcal{C}^{(0)}$.
Compute $\mathcal{C} := \cap_{j\in\mathcal{S}\cup\{0\}}\mathcal{C}^{(j)}$
Send $\mathcal{C}$ to committee members in $\mathcal{S}$

Data Combination Phase by Committee Member j in iteration ℓ

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

Data Aggregation Phase by Server in iteration ℓ

for $in = 1, \ldots, L$ **do**
Let $\left\{\mathsf{AUX}_\ell^{(j,in)}\right\}_{j\in\mathcal{S}}, \left\{\mathsf{ct}_{i,\ell}^{(in)}\right\}_{i\in\mathcal{C}}$ be the inputs received by the server with $|\mathcal{S}| \geqslant \mathsf{t}$.
Run $\mathsf{AUX}_\ell^{(in)} \leftarrow \mathsf{OPA.S\text{-}Combine}(\left\{\mathsf{AUX}_\ell^{(j,in)}\right\}_{j\in\mathcal{S}})$
Run $X_\ell^{(in)} \leftarrow \mathsf{OPA.Aggregate}(\mathsf{AUX}_\ell^{(in)}, \left\{\mathsf{ct}_{i,\ell}^{(in)}\right\}_{i\in\mathcal{C}})$

Figure 10: Our Construction of Two Round Secure Aggregation protocol with m committee members and t being the threshold for reconstruction using $(\mathsf{t}, \mathsf{m}, \mathsf{M})$ One-shot Private Aggregation scheme OPA.

Using Packed Secret Sharing. As in the case of SOPA_1, SOPA_2 can be further optimized by using Packed Secret Sharing. We refer the reader to Remark 5 for a discussion on how to employ packed secret sharing of the multiple keys used in an effort to reduce communication.

Using Signatures for Active Security. In the context of Federated Learning, active security of current protocols aims to offer robustness where a server or a committee member could lie about the set of online clients. To mitigate this, one needs to simply add a signature. Each committee member sends a bit string of length n where i-th bit is set to 1 iff client i participating in that iteration. It also signs this message with its signing keys. This is done by every committee member. The server performs the intersection and when communicating to the committee members, also attaches the list of all signatures to ensure that the intersection is done correctly. Finally, the committee members aggregate their information only after verifying there are at least $\mathtt{t}$ valid signatures and the intersection is computed correctly. Among recent contributions, only the work by [11] introduces a resilient secure aggregation protocol that achieves full malicious security. However, it grapples with a substantial round complexity, approximately $\log n$. This situation raises an intriguing and unresolved challenge: devising a fully robust federated learning protocol that not only reduces round complexity but also transitions to a non-interactive framework remains an open problem.