

Lighthouse: Single-Server Secure Aggregation with $O(1)$ Server-Committee Communication at Scale

Sanjam Garg¹, Alireza Kavousi², Dimitris Kolonelos¹, Erkan Tairi¹, and Zhipeng Wang³

¹University of California, Berkeley

²University College London

³The University of Manchester

Abstract

Secure aggregation is a core primitive for privacy-preserving federated learning, enabling a server to compute aggregates of client updates without learning individual inputs. Recent protocols have explored committee-based designs to reduce client overhead and tolerate weakly connected participants. However, existing approaches still incur communication and computation costs that scale with the number of clients and/or the size of model updates. This becomes a serious bottleneck in interaction between the server and the committee, given that model updates are high-dimensional and the committee is a small set of clients.

We present Lighthouse, a new secure aggregation protocol that supports one-shot client communication and achieves constant committee computation and communication overhead with the server, independent of both the number of clients and the size of the input vector. Our protocol attains the best-known round complexity of two rounds, matching OPA (CRYPTO 2025) and TACITA (ePrint 2025) and improving upon Flamingo (IEEE S&P 2023) and Willow (CRYPTO 2025). Our core technical contribution is a novel application of recent advances in batched threshold encryption, which enables succinct server-committee interaction while preserving security and correctness. Beyond asymptotic improvements over prior works, Lighthouse yields substantial concrete efficiency gains: For an aggregation with 1024 clients, we reduce server-to-committee communication by over $100\times$ and committee-to-server communication by over $300\times$ compared to Flamingo and Willow. Also, we present an extension that supports dynamic client participation, a critical requirement for practical deployments at scale, while preserving the asymptotic and concrete efficiency of the static protocol for clients.

Contents

1	Introduction	3
1.1	Our Contributions	4
1.2	Technical Overview	6
2	Preliminaries	9
2.1	Basic Notation	9
2.2	Cryptographic Building Blocks	9
2.3	Succinct Non-interactive Argument of Knowledge	11
2.4	Threshold Batched Identity-Based Key Encapsulation Mechanism	11
3	Setting and Threat Model	13
3.1	Properties and Functionality	15
4	Secure Aggregation with $O(1)$ Server-Committee Communication	16
4.1	Our Threshold Batched Identity-Based Key Encapsulation Mechanism	17
4.2	Lighthouse: Our Secure Aggregation Protocol	19
4.3	Security Analysis	21
4.4	Robustness of Pairwise Masking Against Input Manipulation	27
4.5	Robustness Against Malicious Labeling in and Across Iterations	27
5	Secure Aggregation with Dynamic Participation	28
5.1	Construction	29
5.2	Security Analysis	31
6	Performance Evaluation	32
6.1	Experiment Setup	32
6.2	Communication Costs	33
6.3	Computation Costs	34
7	Related Work	35
A	Oversight in Prior Constructions	42
B	The Flamingo Protocol	43
C	Succinct Cross-checking Proof	46

1 Introduction

Federated learning (FL) has become an increasingly prominent approach for training machine-learning models across large populations of devices without centralizing raw data [MMR⁺17]. By allowing clients to compute local updates and transmit only their model contributions to a centralized server for aggregation, FL mitigates many privacy and data-governance concerns. However, even the disclosure of individual model updates can leak sensitive information [MSDS19, KMA⁺21]. Secure aggregation [BIK⁺17] has emerged as a core cryptographic building block for FL, enabling a powerful server to recover only an aggregate of client inputs without learning any individual local input. FL systems typically involve tens of thousands of clients, extremely high-dimensional vectors (e.g., model weights), and hundreds of training rounds. Moreover, participation is inherently asynchronous, as it involves unstable clients such as mobile devices: Devices may join at unpredictable times, possess limited compute and bandwidth resources, or disconnect occasionally.

Designing practical secure aggregation protocols for mobile devices presents a unique set of conflicting constraints. First, the protocol must be concretely and asymptotically *efficient*: Client devices are resource-constrained, often communicating over metered or unstable connections, necessitating low communication and computation overhead. Second, it must be robust to *dropouts*: In realistic FL deployments involving thousands of devices, a significant fraction of clients may disconnect mid-protocol due to battery loss or network failure. A practical protocol must guarantee that the aggregation completes successfully despite these failures, without restarting the round or discarding the contributions of surviving clients. Finally, the system should ideally support *dynamic participation*, allowing clients to join or leave the system fluidly across iterations without cumbersome registration phases or group coordination.

Protocols that require tight synchronization or multiple rounds of client interaction incur high latency and are fragile to stragglers and dropouts; in practice, even a few extra communication rounds can dominate end-to-end training time [BIK⁺17, BBG⁺20]. Consequently, there has been growing interest in *one-shot* secure aggregation protocols, where each client sends a single message per aggregation and can immediately go offline. However, in non-interactive settings, a malicious server may attempt to recompute aggregates over carefully chosen overlapping subsets and infer individual inputs via differencing or replay-style attacks. More broadly, fundamental results show that removing interaction renders any secure computation insecure, enabling malleability and recombination attacks [HLP11]. Recent secure aggregation protocols explicitly observe this gap and demonstrate concrete subset-style attacks in the single-server setting [BBG⁺20].

Committee-based single-server secure aggregation. An emerging approach is to introduce a small *committee* of auxiliary clients to assist in aggregation [MWA⁺23]. From a security perspective, the committee acts as a cryptographic gatekeeper: The server can recover an aggregation result only with the committee’s participation, binding each aggregation to a single execution and preventing subset and replay-style attacks. Despite this progress, existing committee-based protocols have notable limitations. In prior protocols, committee members incur linear computation and communication costs of $O(n)$ or $O(\ell)$, with n and ℓ being the number of clients and input vector dimension respectively [MWA⁺23, vKLMS25, KP25], making them a bottleneck in large federated learning workloads. Moreover, other protocols require per-iteration committee interaction [BCGL⁺25] or assume a static set of participating clients [MWA⁺23], which limits their applicability in dynamic, long-running systems.

While these protocols successfully keep regular clients lightweight and one-shot, they shift a substantial burden onto a subset of participants, the committee members, who now should dedicate larger amount of resources, i.e., $O(n)$ or $O(\ell)$ in communication and computation. This undermines the intended design principle of secure aggregation that the clients should be extremely lightweight.

1.1 Our Contributions

In this work, we present a construction that improves upon the best-known prior committee-based single-server secure aggregation protocols. Our construction Lighthouse builds on the pairwise-masking paradigm of [BBG⁺20, MWA⁺23] and simultaneously offers properties required for practical deployment, including (i) optimal cost of $O(1)$ for the server-committee communication, and for the committee computation, independent of the number of clients, dropouts, and size of the input vector, (2) efficiently tolerating high-dimension input vector and large input domain, (3) supporting dynamic client participation without requiring per-iteration committee interaction. In a nutshell:

- **Constant-cost committee-based secure aggregation.** We present a secure aggregation construction with *optimal* $O(1)$ communication between the server and the committee, and constant $O(1)$ computation per committee member, independent of both the number of participating clients n and the input vector dimension ℓ . A full aggregation iteration terminates in just two round trips, matching the best-known result of [KP25, MLLP25]. In contrast to prior committee-based protocols, the committee is asymptotically and concretely lightweight. In particular, Lighthouse improves upon Flamingo [MWA⁺23], which operates under the same system model and threat assumptions, by significantly reducing the costs incurred by committee members. Our construction builds on the recent advances on efficient witness encryption [GKPW24, GHK⁺25] and, in particular, on batched threshold encryption techniques [CGPP24, AFP25, CGPW25] to amortize committee work across clients per iteration.
- **Efficient support for dynamic client participation.** We extend Lighthouse to support *dynamic client participation*, allowing clients to join aggregation rounds ad hoc without prior coordination. We consider a *hybrid participation model*, where an initial pool of clients participates in a static mode, while additional clients may freely join dynamically. The extended protocol preserves the asymptotic and concrete overheads of the static construction for regular clients, who remain one-shot and incur no additional interaction. On the committee side, computation scales only with the number of newly joined clients n' , and communication is linear in the input dimension ℓ . Crucially, this communication can be piggybacked on the transmission of input vectors for aggregation (since committee clients may also participate in the aggregation), yielding overhead that is essentially optimal and matches the baseline cost of sending data.
- **Simulation security.** We formalize the security of our construction in a (stand-alone) simulation-based setting and provide a complete security proof against malicious adversaries with static corruption in Section 4.3. Additionally, we uncovered some mistakes in the security proofs of prior works, which invalidate their theorem statements (see Appendix A).
- **Benchmarking.** Through detailed microbenchmarks and full system experiments, we demonstrate that our secure aggregation protocol significantly outperforms the state of the art. Our results demonstrate that the protocol reduces server-to-committee communication by over to $100\times$ and committee-to-server communication by over $300\times$ compared to Flamingo and Willow, all while maintaining constant-time committee cost for committee.

In Table 1, we compare the asymptotic cost of Lighthouse with prior committee-based works which have similar properties such as being single server, supporting large input dimension and domain, and with no per-client setup.

Scheme	Client		Server		Rounds
	Comm.	Comp.	Comm.	Comp.	
Flamingo [MWA ⁺ 23]	Regular: $\ell + D$ Decryptor: $n + D$	Regular: $\ell + D$ Decryptor: $n + D$	$\ell n + Dn$	$\ell n + Dn$	3
LERNA [LLPT23]	Regular: ℓ Decryptor: $\ell + n$	Regular: ℓ Decryptor: $\ell + n$	ℓn	ℓn	3
LISA [vKLMS25]	Regular: ℓ Decryptor: ℓ	Regular: $\ell + D$ Decryptor: $\ell + n$	ℓn	ℓn	3-6
Willow [†] [BCGL ⁺ 25]	Regular: $\ell \log n$ Decryptor: $\log n + D$	Regular: $\ell \log n$ Decryptor: $\log n + D$	$\ell n \log n$	$\ell n \log n$	4
OPA [KP25]	Regular: $\ell + D$ Decryptor: n	Regular: $\ell + D$ Decryptor: n	$\ell n + Dn$	$\ell n + \text{exp}$	2
WillowFold [HAG ⁺ 26]	Regular: $\ell + \log n$ Decryptor: $\log n + D$	Regular: $\ell + \log n$ Decryptor: $\log n + D$	$\ell n + n \log n$	$\ell n + n \log n$	4
Lighthouse	Regular: $\ell + D$ Decryptor: $O(1)$	Regular: $\ell + D$ Decryptor: $O(1)$	$\ell n + Dn$	$\ell n + Dn$	2

Table 1: Asymptotic comparison of the communication, computation and round complexities across single-server committee-based protocols for input vector ℓ and n clients. For communication, we consider both send and receive together. We exclude any one-time setup from contributing to the round complexity. For the committee-based protocols we denote the size of the set by $D = O(\lambda)$. exp in OPA denotes a constant number of heavy modular exponentiations. [†]Note that Willow relies on a dual-committee architecture: decryptor and verifier committees. The costs reported here reflect their decryptor committee; there is an $O(n)$ overhead imposed on their verifier committee not considered in this table.

Concurrent and independent work [MLLP25]. A concurrent work by Madathil et al., TACITA [MLLP25], also achieves $O(1)$ server-committee communication using different techniques (silent threshold encryption [GKPW24] and multi-key linearly homomorphic threshold signatures [AP19]), however it differs in client efficiency and support for dynamic participation.

First, TACITA requires a costly $O(N)$, where N is the number of all possible clients that may participate in the protocol (and $N \gg n$), preprocessing phase per iteration that mandates *all* clients (not just committee members) to generate system-sized key material during setup. Second, TACITA is optimized for small input domains, as it requires solving a discrete-logarithm instance over homomorphically aggregated inputs. Although extensions for large domains are proposed using known lattice-based techniques [BCGL⁺25, KP25], their reliance on Ring-LWE-based key-homomorphic encryption severely affects their concrete efficiency. In contrast, Lighthouse supports arbitrarily large inputs without additional overhead, which is crucial for practical FL workloads [ZLWL25, KYR⁺24]. Moreover, TACITA requires clients to send ℓ ciphertexts for an ℓ dimensional vector ([MLLP25, Section 9.1]). While Ring-LWE packing can theoretically amortize costs, this becomes difficult for sparse vectors or misaligned dimensions. Finally, TACITA does not support dynamic participation, assuming a closed universe of N clients defined at setup; admitting new clients requires system-wide re-initialization. By contrast, we provide an extension enabling open and dynamic participation, allowing clients to join without registration or a new setup.

We note that Lighthouse relies on an interactive distributed key generation (DKG) [GJKR99] procedure for the committee, while TACITA’s committee works with a non-interactive preprocessing (in the silent setup model [GJM⁺24]). However, our DKG is a one-time operation restricted solely to a small committee (possibly refreshed after many iterations), while in TACITA there is an $O(N)$ setup overhead for both the clients and the committee. Furthermore, we believe that our protocol could similarly be adapted to support

a silent setup by leveraging recent techniques in [GWWW25].

1.2 Technical Overview

Lighthouse is designed to address the scalability and security challenges of secure aggregation in multi-iteration single-server model. We build upon the established pairwise-masking paradigm and introduce a new *threshold decryption* mechanism to minimize the communication and computation overhead of the committee.

Secure aggregation with pairwise masking. Our protocol builds upon the pairwise-masking framework for secure aggregation, established by Bonawitz et al. [BIK⁺17] and optimized by Ma et al. with Flamingo [MWA⁺23]. In this paradigm, the privacy of individual client inputs is guaranteed via an efficient information-theoretic masking. Formally, each client i holds a private vector x_i . To participate in the aggregation, the client masks this input by adding a set of pairwise random values shared with a subset of other peers (neighbors or committee). For every neighbor j , the pair derives a shared secret seed $r_{i,j}$. This typically happens via Diffie-Hellman key exchange in a non-interactive manner (assuming an established public key infrastructure). Crucially, the protocol enforces an additive inverse relationship where client i adds $r_{i,j}$ while client j subtracts it. However, pairwise masking alone is insufficient to guarantee privacy, particularly when the system must accommodate delayed reporting or client dropouts. If the server was to acquire a client’s pairwise secrets—a necessary step to neutralize the noise left by offline neighbors—it could algebraically strip the masking layer and expose the private input. To mitigate this, each client i further includes a secondary individual mask m_i (or self-mask) derived from a private seed. The masked vector y_i computed by client i is thus:

$$y_i = x_i + m_i + \sum_{j \in \mathcal{N}(i), i < j} r_{i,j} - \sum_{j \in \mathcal{N}(i), i > j} r_{i,j}$$

When the server computes the sum of the reported vectors from the set of online clients $\mathcal{U}$, the pairwise masks do not all vanish. While masks between two online clients ($i, j \in \mathcal{U}$) cancel out perfectly, the masks associated with edges between an online client and a dropout ($j \notin \mathcal{U}$) remain as residual noise:

$$\sum_{i \in \mathcal{U}} y_i = \sum_{i \in \mathcal{U}} x_i + \sum_{i \in \mathcal{U}} m_i + \underbrace{\sum_{i \in \mathcal{U}, j \notin \mathcal{U}} \text{sign}(i, j) \cdot r_{i,j}}_{\text{residual pairwise masks}}.$$

This residual term acts as a cryptographic lock on the aggregate. The server cannot compute the final result $\sum x_i$ solely by summing the inputs; it must actively engage the committee to (1) reconstruct and remove the residual pairwise masks corresponding to the set of dropouts, and (2) remove the individual masks $\sum m_i$ for the surviving clients $\mathcal{U}$. This duality creates the core security challenge: The system must ensure the server can remove the pairwise masks of dropouts (to clean the sum) but never the pairwise masks of surviving clients (which would expose their inputs). In other words, the protocol must enforce an exclusivity to ensure server learns the individual mask only for online clients and the pairwise masks only for offline clients.

Prior works handle this by letting the clients compute a threshold encryption [DF90] per mask and send them to the server. That is, each client, together with their contribution, sends threshold encryptions of all their pairwise and individual masks. After receiving the clients’ contributions, the server forwards to the committee an online-offline bit vector $\mathbf{b} = (b_1, \dots, b_n)$ corresponding to the clients’ status. Each

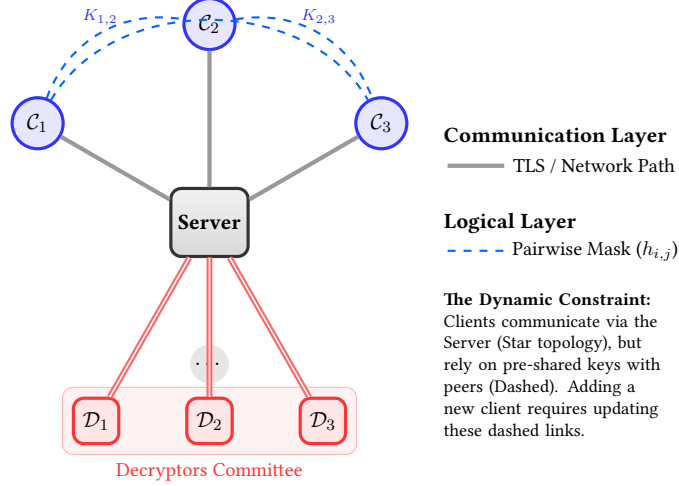


Figure 1: The communication structure of pairwise-masking secure aggregation

committee member signs the vector $\mathbf{b}$ and sends it to the server and the server sends back to the committee all the signatures to reach agreement on the status vector $\mathbf{b}$, assuming an strict honest majority of 2/3. After that, the server sends all the ciphertexts to the committee for decryption. Each committee member answers with the proper (partial) decryptions. This mechanism manually enforces mutual exclusivity—each committee member based on $\mathbf{b}$ manually chooses which pairwise and individual masks to decrypt and which not—which immediately incurs an $O(n)$ communication and computation cost.

We present a visualization of pairwise-masking paradigm in Figure 1. For the sake of completeness, we provide a detailed description of Flamingo [MWA⁺23] in Appendix B.

Ideal solution. Our dream goal would be a cryptographic mechanism that provides “automatic” mutual exclusivity with succinct communication. That is, instead of sending the full online-offline status vector for signing, the server would send to the committee a concise representation of the status and the committee answers concisely such that if $b_i = 1$ (online) the individual mask of client i is “automatically” decrypted and if $b_i = 0$ the corresponding pairwise-masks are decrypted.

This goal would be immediately fulfilled using witness encryption (WE) [GGSW13]. WE is a cryptographic primitive that allows one to encrypt a message with respect to an NP statement such that anyone holding the corresponding witness can decrypt. This would give an $O(1)$ committee as follows:

1. The clients WE-encrypt their masks such that the individual mask can be decrypted iff (i) there is a commitment $\text{cm}_{\mathbf{b}}$ signed by the 2/3 of the committee and (ii) $b_i = 1$. Similarly for the pairwise masks (but instead $r_{j,i}$ ’s are decrypted iff $b_i = 0$).
2. The server sends to the committee a succinct commitment of the status vector $\text{cm}_{\mathbf{b}} = \text{Com}(\mathbf{b})$.
3. The committee signs the commitment $\text{cm}_{\mathbf{b}}$.

Unfortunately, WE has been proven to be an extremely elusive cryptographic primitive to obtain, to-date lacking an efficient realization for general NP languages.

Our core observation. On the core of our protocol is the observation that the above “ideal solution” can be realized efficiently, leveraging recent developments in advanced encryption schemes.

Recent advances in witness encryption [GKPW24, GHK⁺25] have shown that it is possible to construct efficient witness encryption schemes for special-purpose languages. In particular, they show efficient encryption schemes with respect to the KZG polynomial commitment [KZG10] and its vector commitment counterpart [TAB⁺20]. This gives us the following candidate solution: (1) Create a KZG vector commitment cm_b to b , (2) the committee receives cm_b and signs it, (3) given enough signatures the server can decrypt the proper ciphertexts according to the committed status vector b .

In fact, this approach has been leveraged in a recent line of work on batched threshold encryption (BTE) [CGPP24, AFP25, BFOQ25].¹ BTE allows a set of decryptors to decrypt a batch of n ciphertexts at once, by communicating only $O(1)$ partial decryption information (instead of $O(n)$). We build on these works to allow for threshold decryption of many ciphertexts with $O(1)$ communication between the committee and server. Technically in BTE:²

1. Each client encrypts their messages under a unique index i .
2. A server computes and sends to the decryptors a succinct digest cm of the indexes of the batch $\mathcal{I} = \{i_1, \dots, i_n\}$.
3. Each decryptor sends back a succinct partial decryption on cm_b .
4. The server, having enough partial decryptions, can decrypt all ciphertexts in $\mathcal{I}$.

One-of-two decryption. This allows us to achieve $O(1)$ communication, but there is a security caveat: The committee members do not receive the ciphertext or the content of cm_b , and therefore, they cannot verify the type of ciphertexts they are asked to decrypt. In the context of secure aggregation, a malicious server may try to inject both the individual and pairwise mask ciphertexts into cm_b .

For this, we introduce a one-of-two notion for batched (threshold) decryption. For each client i there are two types of ciphertexts, the “online” that encrypts $\{m_i\}$ and is decrypted iff $b_i = 1$ and the “offline” that encrypts $\{r_{j,i}\}_{j \in \mathcal{N}(i)}$ and are decrypted iff $b_i = 0$, respectively. We require that it is impossible for the server to decrypt both, for any adversarial cm of the malicious server’s choice. That is, the server chooses which secret to learn for each client by fixing the value b_i in the digest cm_b , and the committee does not know which clients are online or offline, as they blindly sign the digest, but they know that only one of the two sets can be decrypted. Our cryptographic encryption/decryption procedure here is reminiscent to laconic oblivious transfer [CDG⁺17], in that it parallels the security properties of oblivious transfer but with succinct communication. We refer to 4.1 for the details of our construction.

Finally, we note that this blind authorization introduces a critical challenge. As shown in Flamingo, preserving privacy for all clients requires the committee to enforce specific graph-dependent constraints; that the set of online clients satisfies a minimum quorum (threshold) and maintains connectivity within the logical communication graph. In Flamingo, verifying these conditions necessitates transmitting the full status vector b to the committee. To sidestep this conflict and thus achieve a strictly constant $O(1)$ communication, we augment our solution with succinct verification. Instead of revealing b , the server, using a succinct non-interactive argument of knowledge (SNARK), generates a succinct proof π that ensures that the underlying status vector b committed in cm_b satisfies all necessary security predicates.

Dynamic client participation. We propose a protocol variant to accommodate dynamic client participation by utilizing a client-to-committee zero-sum masking architecture. Each client u_i masks their input x_i by XORing it with pairwise keys derived from the fixed committee set, ensuring that mask terms are

¹ Also similarly in [GWWW25] but diverging from KZG-based techniques.

² We describe an identity-based variant of BTE from [AFP25]. For the sake of this overview we slightly abuse the terminology calling it BTE. We refer to 2.4 for more details.

self-canceling given full committee participation. This design minimizes overhead, ensuring that committee computation scales only with the number of new clients while the client overhead remains intact compared to the static model. To guarantee liveness, we employ our aforementioned batched threshold encryption scheme, which aggregates mask recovery costs into a single batch to achieve a constant $O(1)$ recovery cost regardless of the number of client or committee dropouts. Also, thanks to the committee-centric commitment—fixing the committee configuration rather than the client set—the unmasking procedure only depends on committee member status rather than that of the client set. This would allow new clients to join openly without triggering a global re-setup or incurring the linear overhead of maintaining a global status commitment as in [MLLP25].

Security with selective opening. We show that our protocol achieves the desired security and privacy properties by formally proving the realization of our ideal functionality (Figure 2) even in the presence of a malicious adversary that can control the server and a subset of the clients and committee members (decryptors). However, proving this realization turns out to be highly non-trivial due to the following observation: In committee-based secure aggregation protocols, a malicious server receives the clients’ ciphertexts during the reporting phase, and only at a later phase decides which of these ciphertexts will be decrypted. This implies that the notion of challenge ciphertexts (i.e., the ciphertexts that should not be opened) is not fixed upfront, but instead is adaptively decided by the adversary. Hence, this necessitates using an encryption scheme with stronger security, i.e., *selective opening security*³. In fact, this crucial observation was missed by prior works, Flamingo [MWA⁺23] and TACITA [MLLP25], which resulted in mistakes in their security proofs as we detail in Appendix A.

Therefore, in order to achieve selective opening style security, we switch from using IND-CPA secure TB-IBE to using threshold batch identity-based key encapsulation (TB-IB-KEM) and combine it with random oracles. This allows us to program the random oracle during the security proof in order to simulate the honest parties consistently. Our approach can be viewed as an extension of the selective opening secure PKE construction given in [HJKS15] to the TB-IB setting.

2 Preliminaries

2.1 Basic Notation

We use λ for the security parameter, κ for the statistical security parameter and $\text{negl}(\lambda)$ for a negligible function in λ . We write column vectors with small bold font, e.g. $\mathbf{x} = (x_1, \dots, x_n)^\top$ and matrices in capital bold, e.g. $\mathbf{A} = (a_{ij})_{i,j}$. $x \leftarrow \$ \mathcal{X}$ means that x is being uniformly sampled from a finite set $\mathcal{X}$. By $[n]$ we denote the set $\{1, \dots, n\}$. PPT stands for Probabilistic Polynomial-Time, and every algorithm (including the adversary) is considered to be (stateful) PPT.

2.2 Cryptographic Building Blocks

Bilinear Pairings. A bilinear group, generated as $(\mathbb{F}, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e) \leftarrow \mathcal{BG}(1^\lambda)$, is specified by a field $\mathbb{F}$ of prime order $p = 2^{\Theta(\lambda)}$, three groups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ that we call “pairing” and one random generator g_1, g_2 for each group. We use the *implicit notation*,⁴ i.e., $[x]_s := x \cdot g_s$ and more generally $[\mathbf{A}]_s$ represents a matrix of the corresponding group elements, for $s \in \{1, 2, T\}$. Also, we denote the group operation additively, $[x]_s + [y]_s = [x + y]_s$, for $s \in \{1, 2, t\}$. By

³A selective opening secure encryption scheme guarantees that even if an adversary sees a batch of ciphertexts and then is allowed to “open” a subset of them (revealing the plaintext), the unopened ciphertexts still hide their plaintexts.

⁴We also make use of pairing-free groups; whenever we are in a pairing-free group we use the common multiplicative notation g^x to highlight the difference.

$[x]_1 \circ [y]_2 = [y]_2 \circ [x]_1 = [x \cdot y]_T$, we denote the pairing $e([x]_1, [y]_2)$. For brevity, we will omit writing explicitly the bilinear group description in the algorithms' inputs.

Lagrange Interpolation. Let $\omega \in \mathbb{Z}_p$ be an M -th primitive root of unity, i.e. $\omega^M = 1$ over a field $\mathbb{F}$. ω generates the multiplicative subgroup of roots of unity $\{\omega, \omega^2, \dots, \omega^M\}$. Then $\mathcal{L}_1(x), \mathcal{L}_2(x), \dots, \mathcal{L}_\ell(x)$ is the Lagrange basis of the subgroup. That is, $\mathcal{L}_i$ is the unique degree- $(M - 1)$ polynomial defined by: $\mathcal{L}_i(\omega^j)$ is 1 when $i = j$ and 0 when $i \neq j$. Concretely, $\mathcal{L}_i(X) = \prod_{j=1, j \neq i}^M \frac{X - \omega^j}{\omega^i - \omega^j}$. We denote by $Z(X) = \prod_{i=1}^M (X - \omega^i)$ the vanishing polynomial of the subgroup. Since we are in a multiplicative subgroup, $Z(X) = X^\ell - 1$.

Pseudorandom Generator (PRG). Let $\text{PRG} : \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\ell(\lambda)}$ be a deterministic polynomial-time algorithm, where $\ell(\lambda) > \lambda$ represents the expansion factor. We define it as a secure pseudorandom generator (PRG) if its output is computationally indistinguishable from a truly random string of the same length. Formally, for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the distinguishing advantage must be negligible in the security parameter λ . That is, $|\Pr[\mathcal{A}(\text{PRG}(s)) = 1] - \Pr[\mathcal{A}(r) = 1]| \leq \text{negl}(\lambda)$, where the probability is taken over the uniform choice of the seed $s \xleftarrow{\$} \{0, 1\}^\lambda$ and the uniform random string $r \xleftarrow{\$} \{0, 1\}^{\ell(\lambda)}$.

Pseudorandom Function (PRF). Let $\text{PRF} : \{0, 1\}^\lambda \times \{0, 1\}^n \rightarrow \{0, 1\}^m$ be an efficient deterministic algorithm, where the first input is the key and the second is the input message. We say PRF is a pseudorandom function if no PPT adversary $\mathcal{A}$ can distinguish the input-output behavior of $F_k(\cdot)$ (instantiated with a hidden key k) from that of a truly random function f . Specifically, we require that $|\Pr[\mathcal{A}^{\text{PRF}_k(\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1]| \leq \text{negl}(\lambda)$. Here, the key $k \xleftarrow{\$} \{0, 1\}^\lambda$ is sampled uniformly, and f is sampled uniformly from the set of all functions mapping $\{0, 1\}^n$ to $\{0, 1\}^m$.

Diffie–Hellman Key Exchange. Let $\mathbb{G}$ be a cyclic group of prime order q with generator g . In the Diffie–Hellman key exchange protocol, two parties choose random exponents $a, b \xleftarrow{\$} \mathbb{Z}_q$, publish g^a and g^b , and compute the shared key g^{ab} . Security against passive eavesdroppers relies on the Decisional Diffie–Hellman (DDH) assumption, which guarantees that the tuple (g, g^a, g^b, g^{ab}) is computationally indistinguishable from (g, g^a, g^b, g^c) for a random $c \xleftarrow{\$} \mathbb{Z}_q$, and hence that the derived key g^{ab} is indistinguishable from a random group element.

Random Oracle Model. In the random oracle model (ROM), parties are given oracle access to some function H that is sampled uniformly at random from the space of all functions $H : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are finite non-empty sets. That is, parties can query their oracle on an input $x \in \mathcal{X}$ and receive in return $H(x) \in \mathcal{Y}$. When proving the security of a protocol in the ROM, the simulator $\mathcal{S}$ is able to program the oracle, observing queries made by the adversary and simulating responses.

KZG Commitment. A polynomial commitment scheme [KZG10] allows one to commit to a polynomial $f(X) \in \mathbb{F}[X]$ of degree at most d , with the ability to later provide succinct proofs for evaluations of $f(x)$. A widely used efficient instantiation of a polynomial commitment scheme is KZG commitment [KZG10], that works over Bilinear Groups. The setup phase generates a common reference string (CRS) containing $\{[\tau_i]_1\}_{i=0}^d, [\tau]_2$ for a secret trapdoor $\tau \xleftarrow{\$} \mathbb{F}$. A commitment to a polynomial $f(x)$ is computed as a single group element $\text{cm} = [f(\tau)]_1$, which is efficiently computable in $O(n)$ time given the powers $[\tau^i]_1$ from the CRS. To open the commitment at an evaluation point z with value $y = f(z)$, the prover computes a proof element $\pi = [Q(\tau)]_1$, where $Q(x) = \frac{f(X) - y}{X - z}$ is the quotient polynomial. Verification is performed

by checking the pairing equation $(\text{cm} - [y]_1) \circ [1]_2 = \pi \circ ([\tau]_2 - [z]_2)$, which confirms the divisibility relation in the exponent. This construction is notable for having constant-size commitments and proofs, independent of the polynomial degree.

We will make use of KZG as a Vector Commitment [CF13], leveraging the Lagrange interpolation, as shown in [TAB⁺20]. That is, one can commit to a vector $(v_1, \dots, v_M) \in \mathbb{F}^M$ by first interpolating the points $(\omega^1, v_1), (\omega^2, v_2), \dots, (\omega^M, v_M)$ as $v(X) = \sum_{i=1}^M v_i \mathcal{L}_i(X)$ and then committing to $v(X)$. Consequently, verification for position i gets $(\text{cm} - [v_i]_1) \circ [1]_2 = \pi_i \circ ([\tau]_2 - [\omega^i]_2)$, where the proof is $\pi_i = [Q_i(\tau)]_1$ is the corresponding quotient. For KZG over multiplicative subgroups all proofs can be computed in $O(M \log M)$ (instead of the naive $O(M^2)$) time [FK23, ZOZ25]. For security, it satisfies position-binding, stating that the commitment cannot be opened to two different values at any position $i \in \{1, \dots, M\}$.

2.3 Succinct Non-interactive Argument of Knowledge

A proof system enables a prover P to convince a verifier V about some statement u such that $\exists w : (u; w) \in \mathcal{R}$, where w is the corresponding witness and $\mathcal{R}$ is a polynomial-time decidable relation. A proof of knowledge system further convinces the verifier that not only the witness exists, but also the prover knows it. A proof system is an argument when it holds for a computationally bounded prover under certain computationally hard assumption.

Definition 1 (SNARK). *A succinct (non-interactive) argument system $\text{SNARK} = (\text{Setup}, \text{Prove}, \text{Vf})$ for a relation $\mathcal{R}$ satisfies the following properties:*

- **Completeness.** *Given a true statement u from the relation $\mathcal{R}$, a honest prover with a valid witness w should convince the verifier. More formally, for all $\lambda \in \mathbb{N}$ and for all $(u, w) \in \mathcal{R}$ it holds that*

$$\Pr \left[\text{Vf}(\text{crs}, u, \pi) = 1 \mid \begin{array}{l} \text{crs} \leftarrow \text{Setup}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \text{Prove}(\text{crs}, u, w) \end{array} \right] = 1.$$

- **Knowledge Soundness.** *For every PPT adversarial prover $\mathcal{A}$, there exists a PPT extractor $\text{Ext}_{\mathcal{A}}$, such that for every $\lambda \in \mathbb{N}$, it holds that*

$$\Pr \left[\text{Vf}(\text{crs}, u, \pi) = 1 \wedge (u, w) \notin \mathcal{R} \mid \begin{array}{l} \text{crs} \leftarrow \text{Setup}(1^\lambda, \mathcal{R}) \\ (u, \pi) \leftarrow \mathcal{A}(\text{crs}) \\ w \leftarrow \text{Ext}_{\mathcal{A}}(\text{crs}, u, \pi) \end{array} \right] \leq \text{negl}(\lambda).$$

- **Succinctness.** *A non-interactive argument for a relation $\mathcal{R}$ is succinct if the proof size satisfies $|\pi| \leq \text{poly}(\lambda) \cdot \text{polylog}(|\mathcal{R}|)$, and the verifier runs in time $\leq \text{poly}(\lambda, |u|) \cdot \text{polylog}(|\mathcal{R}|)$. In the preprocessing (circuit-specific) CRS model, we additionally allow $|\text{crs}| \leq \text{poly}(\lambda, |\mathcal{R}|)$.*

2.4 Threshold Batched Identity-Based Key Encapsulation Mechanism

Batched Threshold Encryption (BTE), recently introduced by Choudhuri et al. [CGPP24], generalizes classic threshold encryption [DF90] where a ciphertext can be decrypted when $(f + 1)$ -out-of- L members of a decryption committee provides a (partial) decryption. BTE allows one to decrypt a batch of B ciphertexts with only a constant-sized $O(1)$ partial-decryption information from each decryptor. This is in contrast to the traditional threshold encryption schemes that would render a linear cost proportional to the batch size B .

Agarwal et al. [AFP25] introduced the generalization of threshold batched identity-based encryption (TB-IBE), where each ciphertext is additionally tagged with an identity, namely, the message is encrypted for an identifier id . Then the decryption committee authorizes the decryption of all ciphertexts for a batch of identities $\{id_1, \dots, id_B\}$. Ciphertexts can be decrypted if and only if at least $(f + 1)$ -out-of- L members of the committee vouched for the decryption. In TB-IBE this proceeds in two steps, first the set of identities is compressed to a succinct digest cm that is sent to the committee and then each decryptor, using their secret key, computes a partial decryption pd_i . Having $(f + 1)$ -out-of- L partial decryptions one can decrypt any ciphertext under the any identity in the batch $\{id_1, \dots, id_B\}$. Crucially, many different messages can be encrypted under the same id_i and can all be decrypted.

In both batched threshold encryption and threshold batched identity-based encryption the process is tagged with an epoch identifier.⁵ That is, the ciphertexts and partial decryptions are valid only for a specific epoch. Importantly, this renders the identities in TB-IBE reusable (for different epoch identifiers). In this work we extend the definition of TB-IBE from [AFP25] to a key encapsulation setting.

Definition 2 (Threshold Batched Identity-Based Key Encapsulation Mechanism). *A threshold batched identity-based key encapsulation mechanism (TB-IB-KEM) is a tuple of algorithms $TBIBKEM = (\text{Setup}, \text{KGen}, \text{Encap}, \text{Digest}, \text{BatchPartDec}, \text{Decap})$ such that:*

- $\text{Setup}(1^\lambda, 1^B) \rightarrow \text{crs}$: On input the security parameter λ and a batch size B , outputs a common reference string.
- $\text{KGen}(\text{crs}, L, f) \rightarrow (\text{pk}, (\text{sk}_1, \dots, \text{sk}_L))$: On input the common reference string, a number of decryptors L and a threshold f , outputs the public key pk along with secret keys $(\text{sk}_1, \dots, \text{sk}_L)$ for each party.
- $\text{Encap}(\text{pk}, \text{eid}, id) \rightarrow (k, \text{ct})$: On input a public key pk , an epoch id eid and an identity id , outputs a key k and an encapsulation ct .
- $\text{Digest}(\text{crs}, \{id_1, \dots, id_B\}) \rightarrow \text{cm}$: On input the common reference string crs and a set of B identities $id_1, \dots, id_B$, outputs a digest cm .
- $\text{BatchPartDec}(\text{sk}_i, \text{cm}, \text{eid}) \rightarrow \text{pd}_i$: On input the digest cm , the epoch id eid and the secret key share sk_i , outputs a partial decapsulation pd_i .
- $\text{Decap}(\text{crs}, \text{eid}, (id_1, \dots, id_B), \{\text{pd}_i\}_{i \in S}, id, \text{ct}) \rightarrow k$: On input the crs , the epoch id eid , the batch of B identities $(id_1, \dots, id_B)$ and $f + 1$ partial decryptions $\{\text{pd}_i\}_{i \in S}$ where $S \subseteq [L]$ and $|S| = f + 1$ and an encapsulation ct tagged under an identity id , outputs a key k or $\perp$.

Furthermore it should satisfy:

Correctness. For every $\lambda \in \mathbb{N}$, $B = \text{poly}(\lambda)$, $L \in \mathbb{N}$, $f \in [L - 1]$, $\text{eid} \in \{0, 1\}^*$, for every subset $I := \{id_i\}_{i \in [B]}$ from the identity space, some $id \in I$ and $S \subseteq [L]$ such that $|S| \geq f + 1$:

$$\Pr \left[\begin{array}{c} \tilde{k} = k \\ \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^B) \\ (\text{pk}, (\text{sk}_1, \dots, \text{sk}_L)) \leftarrow \text{KGen}(\text{crs}, L, f) \\ (k, \text{ct}) \leftarrow \text{Encap}(\text{pk}, \text{eid}, id) \\ \text{cm} \leftarrow \text{Digest}(\text{crs}, I) \\ \{\text{pd}_i \leftarrow \text{BatchPartDec}(\text{sk}_i, \text{cm}, \text{eid})\}_{i \in S} \\ \tilde{k} \leftarrow \text{Decap}(\text{crs}, \text{eid}, I, \{\text{pd}_i\}_{i \in S}, id, \text{ct}) \end{array} \right] = 1.$$

⁵This is not the case for all existing realizations, e.g., for [BFOQ25, BCF⁺25, BLT25, ABD⁺25]. In this work, we rely on epoch-dependent (T)B-IBE, a property that we need.

Security. A threshold batched identity-based key encapsulation mechanism TBIBKEM has one-way security in the presence of a plaintext-checking oracle (OW-PCA), if for all $\lambda, B, L, f \in \mathbb{N}$, for every PPT adversary $\mathcal{A}$, for every corruption set $\mathcal{C} \subset [L]$ such that $|\mathcal{C}| \leq f$, there exists a negligible function negl such that the following holds

$$\text{Adv}_{\text{TBIBKEM}}^{\text{ow-pca}} := \Pr \left[\text{Exp}_{\mathcal{A}, \text{TBIBKEM}}^{\text{ow-pca}}(\lambda, B, L, f, \mathcal{C}) = 1 \right] \leq \text{negl}(\lambda),$$

where the experiment $\text{Exp}_{\mathcal{A}, \text{TBIBKEM}}^{\text{ow-pca}}$ is defined as follow

$\text{Exp}_{\mathcal{A}, \text{TBIBKEM}}^{\text{ow-pca}}(\lambda, B, L, f, \mathcal{C})$ <pre> 1: $\mathcal{Q} := \emptyset$ 2: $\text{crs} \leftarrow \text{Setup}(1^\lambda, 1^B)$ 3: $(\text{pk}, (\text{sk}_1, \dots, \text{sk}_L)) \leftarrow \text{KGen}(\text{crs}, L, f)$ 4: $(\text{eid}^*, \text{id}^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\text{PartDec}}, \mathcal{O}_{\text{Check}}}(\text{crs}, \text{pk}, \{\text{sk}_i\}_{i \in \mathcal{C}})$ 5: $(\text{k}, \text{ct}) \leftarrow \text{Encap}(\text{pk}, \text{eid}^*, \text{id}^*)$ 6: $\text{k}' \leftarrow \mathcal{A}^{\mathcal{O}_{\text{PartDec}}, \mathcal{O}_{\text{Check}}}(\text{ct})$ 7: return $\text{k} = \text{k}'$ </pre> $\mathcal{O}_{\text{PartDec}}(\{\text{id}_1, \dots, \text{id}_B\}, \text{eid})$ <pre> 1: if $\text{eid} \in \mathcal{Q} \vee (\text{eid} = \text{eid}^* \wedge \text{id}^* \in \{\text{id}_1, \dots, \text{id}_B\})$ then 2: return $\perp$ 3: $\mathcal{Q} := \mathcal{Q} \cup \{\text{eid}\}$ 4: $\text{cm} \leftarrow \text{Digest}(\text{crs}, \{\text{id}_1, \dots, \text{id}_B\})$ 5: $\{\text{pd}_i \leftarrow \text{BatchPartDec}(\text{sk}_i, \text{cm}, \text{eid})\}_{i \in [L] \setminus \mathcal{C}}$ 6: return $\{\text{pd}_i\}_{i \in [L] \setminus \mathcal{C}}$ </pre> $\mathcal{O}_{\text{Check}}(\text{k}, \text{ct}, \{\text{id}_1, \dots, \text{id}_B\}, \text{id}, \text{eid})$ <pre> 1: $\text{cm} \leftarrow \text{Digest}(\text{crs}, \{\text{id}_1, \dots, \text{id}_B\})$ 2: $\{\text{pd}_i \leftarrow \text{BatchPartDec}(\text{sk}_i, \text{cm}, \text{eid})\}_{i \in [L]}$ 3: $\text{k}' \leftarrow \text{Decap}(\text{crs}, \text{eid}, (\text{id}_1, \dots, \text{id}_B), \{\text{pd}_i\}_{i \in [L]}, \text{id}, \text{ct})$ 4: return $\text{k} = \text{k}'$ </pre>

3 Setting and Threat Model

Roles. There are three main entities in our secure aggregation protocol: clients $\mathcal{C}_1, \dots, \mathcal{C}_N$, server $\mathcal{S}$, and decryptors $\mathcal{D}_1, \dots, \mathcal{D}_L$. At each iteration t of aggregation, the protocol aims at computing the sum of the input vectors $\sum_{i \in S_t} \mathbf{x}_{i,t} \in \mathbb{F}^\ell$ of a sampled population S_t of the clients attached to a unique index $i \in [n_t]$ such that no information about any individual input $\mathbf{x}_i$ is leaked while tolerating a certain fraction of client dropouts. Similar to Flamingo [MWA⁺23], in our base variant we assume that the set of all clients is fixed and a subset of them S_t is chosen to participate in a given iteration t . Decryptors are a set/committee of clients with the particular role of helping the server with computing the final result and also attesting for its honesty in protocol coordination. They can also contribute to each iteration of aggregation by sending their own input vectors. The protocol advances in synchronous rounds initiated by the server: In each round trip, the server multicasts messages to the clients, waits a fixed known interval to accumulate their replies, processes the collected messages, and then initiates the subsequent round. We present a summary

of the notations for our secure aggregation setting in Table 2.

Communication model. Communication between each client and the server takes place over a confidential, authenticated channel and messages sent from clients to other clients are forwarded via the server, and are end-to-end encrypted and authenticated. We do not assume any point-to-point channels between the clients as they might not know each other a priori and consider a star-like communication network where the server coordinates the communication by routing the messages between each pair of clients. Clients are typically devices with limited resources making them vulnerable to dropout and short-lived presence. Thus, we aim for *one-shot* aggregation where each client only speaks once during the aggregation iteration by sending the (masked) input vector to the server and can immediately go offline

Symbol	Description
N	Overall number of clients
t	Index for the iteration number
ℓ	Size of the client's input vector $ x_{i,t} $
δ	Maximum fraction of dropouts
η	Maximum fraction of the corruption
n_t	Number of selected clients in iteration t
S_t	Set of selected clients in iteration t
G_t	Communication graph in iteration t
$\mathcal{S}$	Server
$\mathcal{C}_i$	Client i
$\mathcal{D}_i$	Committee member i
L	Size of decryptors committee

Table 2: Notation summary.

Adversary. We consider a static and maliciously secure adversary who might corrupt the server, clients, and up to a threshold number of decryptors, where the majority remains honest. Similar to previous works [MWA⁺23, MGKP25], we distinguish the event of corruption and drop out, where a client or decryptor fail to carry out their task in a timely manner (i.e., a known time period) at any given iteration of the aggregation. Specifically, we assume that at most a fraction δ of clients and at most δ_D of decryptors fail to participate, while also accounting for the proportions of corrupted regular clients η and corrupted decryptors η_D . To ensure correct and secure secure aggregation, we require that the combined fraction of decryptor dropouts and corrupted decryptors satisfies $\delta_D + \eta_D < \frac{1}{3}$ and for the set of n clients contributing to a given iteration it holds $\delta + \eta < \frac{1}{3}$. Note that the choice of η_D depends on the global parameter η and how we sample the committee members $\{\mathcal{D}_i\}_i$. One possible approach is following the sampling method used in Flamingo [MWA⁺23] which assumes the existence of some public randomness. The dropout rate δ depends on the time chosen for the waiting time and can be adjusted according to the application needs. We refer the reader to [MWA⁺23] for formal analysis of the relation between η_D and η .

Client selection and graph connectivity. We assume that in each aggregation iteration t , the set of participating clients S_t is chosen by a public client-selection algorithm (e.g., Flamingo [MWA⁺23]), allowing each party to locally determine the active clients and committee members. To facilitate pairwise masking among this cohort, the participating clients are mapped to a logical, undirected communication graph $G_t = (V_t, E_t)$, where $V_t = S_t$. For any client $i \in V_t$, their neighborhood is formally defined as $\mathcal{N}_t(i) := \{j \in V_t \mid (i, j) \in E_t\}$. Crucially, to ensure a consistent view without prior peer-to-peer interaction, G_t is generated deterministically by all parties using a public randomness beacon [KWJ24] (or

the setup seed ν) combined with the iteration identifier t , yielding a highly connected pseudo-random graph [MWA⁺23]. Consequently, the server, committee, and clients can independently compute the exact set of neighbors $\mathcal{N}_t(i)$ for any client. Finally, for key management among these discovered peers, we assume a verifiable Public Key Infrastructure (PKI) [MBB⁺15, TBP⁺19, TKPS22, LCG⁺24, HSBB25], ensuring authentic public keys and protecting the one-shot aggregation from man-in-the-middle attacks.

Client participation model. The client participation in our protocol operates primarily in a *registered model*, similar to other schemes like Flamingo [MWA⁺23] and TACITA [MLLP25], where a universe of potential participants is established during a one-time setup phase. This configuration inherently assumes a “closed world” where the participant pool is static; adding new users typically necessitates a system-wide re-initialization. To address this limitation, we present a protocol extension that enables *dynamic, open-world participation*. Our extension allows arbitrary new clients to join an active aggregation round on the fly. We thus consider our client participation model as hybrid offering the best of both worlds: the stability and efficiency of pairwise masking for core users, and the flexibility of ephemeral injection for dynamic participants.

One-time Setup. The system employs a global public key pk , whose corresponding secret key sk is secret-shared among a designated committee of $L = 3f + 1$ decryptors. The protocol tolerates up to f malicious committee members and requires any $2f + 1$ valid partial decryptions, which the server can combine without relying on decryptor-to-decryptor interaction. To generate this threshold key in a server-mediated star topology, we follow Flamingo’s approach by adapting the Distributed Key Generation (DKG) protocol of Gennaro et al. [GJKR99]. This adaptation relaxes the typical broadcast assumptions to handle potential message dropouts caused by the server. Importantly, this is a one-time setup phase executed only between the server and the small set of committee members.

3.1 Properties and Functionality

Intuitively, our secure aggregation protocol satisfies three main properties under aforementioned threat model including *security*, *correctness*, and *dropout resilience*. Security ensures that for any maliciously secure PPT adversary that corrupts server and a minority of decryptors, no information about the honest (online) client’s input is leaked beyond what is implied by the result and any public information. Correctness guarantees that in the event of having clients follow the protocol (i.e. submit well-formed inputs), the server derives the correct sum of the online clients’ inputs. Dropout resilience states for any set of sufficiently large online clients during the protocol execution $\mathcal{X} \subset [n]$, the server should output $\sum_{i \in \mathcal{X}} x_{i,t}$.

There are, however, two cases that are not within the scope of our work. First, we do not aim to ensure correctness against *poisoning attacks* by malicious clients who might submit bogus inputs to cause the final result to be wrong. There is an evolving line of research on techniques to provide input correctness or validation [ABIW22, BGL⁺23, MGKP25]. Second, consistent with other single-server secure aggregation protocols [MWA⁺23, KP25, MLLP25], our protocol achieves security with abort where the adversary can cause the protocol to abort, but non-adaptively and without learning the result. While we guarantee robustness against client dropouts, guaranteed output delivery (GOD) is not achieved in case a large number of dropouts (at least $(1 - \delta)|S_t|$) occur in any given iteration t or the malicious server causes the honest decryptors to abort. Intuitively, achieving GOD against a malicious server is generally infeasible in a star topology, as the server acts as the sole communication hub and can deny service at will. We present the ideal functionality for our secure aggregation in Figure 2.

Parties: a server and a (potentially unbounded) set of clients.

Parameters: corruption rate η , dropout rate δ , number of per-iteration participating clients n .

- *Initial setup.* There is an initial population of clients $U_0 = [N] = \{1, \dots, N\}$. The adversary $\mathcal{A}$ chooses an initial corrupted set $C_0 \subseteq U_0$ such that $|C_0|/|U_0| \leq \eta$. The functionality maintains a current universe U_t of all clients that have appeared up to iteration t , and a current corrupted set $C_t \subseteq U_t$.

- For each iteration $t \in [T]$:

1. *New clients join.* The functionality receives from the environment a set of new client identities $J_t \subset \mathbb{N} \setminus U_{t-1}$. It sets $U_t \leftarrow U_{t-1} \cup J_t$, sends J_t to $\mathcal{A}$, and receives from $\mathcal{A}$ a set of newly corrupted clients $J_t^{\text{cor}} \subseteq J_t$ such that

$$\frac{|C_{t-1} \cup J_t^{\text{cor}}|}{|U_t|} \leq \eta$$

It then sets $C_t \leftarrow C_{t-1} \cup J_t^{\text{cor}}$.

2. *Identifying participants and dropouts.* The functionality receives a sized- n random subset $S_t \subseteq U_t$, a set of corrupted parties C_t (in the static setting $C_t = C_0$), and a dropout set $O_t \subseteq S_t$ such that $|O_t|/|S_t| \leq \delta$ and $|C_t|/|S_t| \leq \eta$. Moreover, it receives a sized- L random subset $D \subseteq [N]$ and a corresponding dropout set $O' \subseteq D$ such that $|O'|/|D| \leq \delta_D$. The functionality sends (S_t, O_t, D, O') to $\mathcal{A}$. Each honest client $i \in S_t \setminus (O_t \cup C_t)$ provides an input vector $\mathbf{x}_{i,t}$ to the functionality.
3. *Adversary chooses active set.* The functionality asks $\mathcal{A}$ for a set $M_t \subseteq S_t \setminus O_t$ and a set $M' \subseteq D \setminus O'$. Let the set of new honest contributors in iteration t be $H_t^{\text{new}} = (M_t \cap J_t) \setminus C_t$. If either

$$|M_t|/|S_t| < 1 - \delta \quad \text{or} \quad |H_t^{\text{new}}| < 2 \quad \text{or} \quad |M'|/|D| < 1 - \delta_D$$

then $\mathcal{F}_{\text{Agg}}$ sends abort to all honest parties and halts. Otherwise, it computes

$$\mathbf{y}_t = \sum_{i \in M_t \cup M' \setminus C_t} \mathbf{x}_{i,t}$$

4. *Output to the server.* Depending on whether the server is corrupted by $\mathcal{A}$:
 - If the server is corrupted, then $\mathcal{F}_{\text{Agg}}$ outputs $\mathbf{y}_t$ to all parties corrupted by $\mathcal{A}$.
 - If the server is not corrupted, then $\mathcal{F}_{\text{Agg}}$ asks $\mathcal{A}$ for a shift vector $\mathbf{a}_t$ and outputs $\mathbf{y}_t + \mathbf{a}_t$ to the server.

Figure 2: Secure aggregation functionality $\mathcal{F}_{\text{Agg}}$. The blue parts are for the extended functionality for dynamic participation.

4 Secure Aggregation with $O(1)$ Server-Committee Communication

In this section we present our core secure aggregation protocol, Lighthouse. We begin by describing a variant of the TB-IB-KEM scheme we use in the protocol and then we proceed to the actual protocol description, its security and its properties.

4.1 Our Threshold Batched Identity-Based Key Encapsulation Mechanism

We make use of a threshold batched identity-based key encapsulation mechanism (TB-IB-KEM) with two additional properties. First, our identities are consecutive, lying in a prespecified interval, that is, the identities in the batch are always $1, \dots, B$. So far this would render the batched property trivial: If the identities are always the same they can be regarded as one identity. The non-triviality stems from the second property; for every identity id (and epoch eid) let two encapsulations and keys $(k_0, \text{ct}_0), (k_1, \text{ct}_1)$, then exactly one of the two keys should be obtained, even when the digest is adversarially chosen. We express these as separate identities $\text{id}||0$ and $\text{id}||1$ and we require that it's not possible that both $\text{id}||0$ and $\text{id}||1$ are included in the batch. The batch now is $(1||b_1, \dots, B||b_B)$, for $b_i \in \{0, 1\}$.

Our construction is a variant of [AFP25], adjusted to the KEM setting. There, the digest of the identities is a KZG commitment to the polynomial $\frac{\prod_{i=1}^B (X - \text{id}_i)}{\prod_{i=1}^B \text{id}_i}$, i.e. a polynomial interpolating the points $(\text{id}_1, 0), \dots, (\text{id}_B, 0)$ (and normalizing such that the constant is 1). In our case we interpolate $\{(1, b_1), \dots, (B, b_B)\}$, where $b_i \in \{0, 1\}$. This means that if $(i, 0)$ is interpolated then the $i||0$ -tagged KEM keys are obtained and if $(i, 1)$ is interpolated the $i||1$ -tagged KEM keys are obtained. Concretely the digest is a KZG commitment: $\text{cm} = [\sum_{i=1}^B b_i \mathcal{L}_i(\tau)]_1$.

Construction. Our threshold batched identity-based key encapsulation mechanism TBIBKEM works as follows:

- **Setup** $(1^\lambda, 1^B)$: Sample $\tau \leftarrow \mathbb{Z}_p$ and output:

$$\text{crs} = ([\tau^1]_1, \dots, [\tau^B]_1, [\tau^1]_2, \dots, [\tau^B]_2).$$

- **KGen** (crs, L, f) : Sample $\text{sk} \leftarrow \mathbb{Z}_p$ and compute (f, L) -shares of sk . Output:

$$\text{pk} = [\text{sk}]_2, \quad (\text{sk}_1, \dots, \text{sk}_L) = (\langle \text{sk} \rangle_1, \dots, \langle \text{sk} \rangle_n)$$

- **Encap** $(\text{pk}, \text{eid}, i||b)$: Let

$$\mathbf{A} = \begin{pmatrix} [1]_2 & [\omega^i - \tau] & 0 \\ \text{pk} & 0 & [1]_2 \end{pmatrix}, \quad \mathbf{k} = \begin{pmatrix} [b]_T \\ H(\text{eid}) \circ \text{pk} \end{pmatrix}.$$

Sample $\mathbf{r} \leftarrow \mathbb{Z}_p^2$ and output

$$\text{ct} = \mathbf{r}^\top \mathbf{A}, \quad \mathbf{k} = \mathbf{r}^\top \mathbf{k}$$

- **Digest** $(\text{crs}, \{1||b_1, \dots, B||b_B\})$: Output:

$$\text{cm} = \left[\sum_{i=1}^B b_i \mathcal{L}_i(\tau) \right]_1$$

- **BatchPartDec** $(\text{sk}_j, \text{cm}, \text{eid}) \rightarrow \text{pd}_j$: Output:

$$\text{pd}_j = \text{sk}_j \cdot (H(\text{eid}) - \text{cm})$$

- **Decap** $(\text{crs}, \text{eid}, (1||b_1, \dots, B||b_B), \{\text{pd}_i\}_{i \in S}, i||b_i, \text{ct})$: If $i > B$ or $b_i \notin \{0, 1\}$ or $|S| \leq f$ output $m = \perp$. Let the polynomial $b(X) = \sum_{i=1}^B b_i \mathcal{L}_i(X)$. Compute the KZG commitment $\text{cm} = [b(\tau)]_1$. Compute the KZG opening proof π_i , corresponding to the polynomial $b(X)$, at the point (ω^i, b_i) , i.e., π_i is a proof for $b(\omega^i) = b_i$. Use the $|S| > f$ partial signatures pd_j on $H(\text{eid}) - \text{cm}$ to compute the full signature σ (by reconstructing in the exponent). Let $\mathbf{w} = (\text{cm}, \pi_i, \sigma)$, output:

$$\mathbf{k} = \text{ct} \circ \mathbf{w}^\top$$

Theorem 1. *The above construction, TBIBKEM, is a OW-PCA secure threshold batched identity-based key encapsulation mechanism in the generic group model.*

The theorem follows almost directly from [AFP25]. There the authors proved the stronger notion of indistinguishability (for the encryption variant of the above) which implies the one-wayness property that we require (see Definition 2). We refer to [KL07] for a relation between the two notions.

One-of-Two security. Furthermore, for the security proof of our secure aggregation protocol (given in Theorem 3) we require an additional property that we call *one-of-two security*. Intuitively, it says that an adversary cannot learn the KEM key corresponding to both identities $i||0$ and $i||1$ at the same epoch, even if they choose the digest cm . We note that the OW-PCA property does not capture this property.

Definition 3 (One-of-two Security). *A threshold batched identity-based key encapsulation mechanism TBIBKEM satisfies one-of-two security, if for all $\lambda, B, L \in \mathbb{N}$, $f \in [L - 1]$, for every $i \in [B]$, for every $\text{eid} \in \{0, 1\}^*$, and for every subset $S \subseteq [L]$ such that $|S| \geq f + 1$ and for every PPT adversary $\mathcal{A}$, there exists a negligible function negl such that it holds*

$$\text{Adv}_{\text{TBIBKEM}}^{\text{one-of-two}} := \Pr \left[\begin{array}{c} \tilde{k}_i^0 = k_i^0 \\ \wedge \tilde{k}_i^1 = k_i^1 \end{array} \mid \begin{array}{l} \text{crs} \leftarrow \text{Setup}(1^\lambda, 1^B) \\ (\text{pk}, (\text{sk}_1, \dots, \text{sk}_L)) \leftarrow \text{KGen}(\text{crs}, L, f) \\ \{ (k_i^b, \text{ct}_i^b) \leftarrow \text{Encap}(\text{pk}, \text{eid}, i||b) \}_{b \in \{0,1\}} \\ (\text{cm}, I) \leftarrow \mathcal{A}(\text{crs}, \text{pk}, \{ \text{sk}_j \}_{j \in [L]}, \{ \text{ct}_i^b \}_{b \in \{0,1\}}) \\ \{ \text{pd}_k \leftarrow \text{BatchPartDec}(\text{sk}_i, \text{cm}, \text{eid}) \}_{k \in S} \\ \tilde{k}_i^0 \leftarrow \text{Decap}(\text{crs}, \text{eid}, I, \{ \text{pd}_k \}_{k \in S}, i||0, \text{ct}_i^0) \\ \tilde{k}_i^1 \leftarrow \text{Decap}(\text{crs}, \text{eid}, I, \{ \text{pd}_k \}_{k \in S}, i||1, \text{ct}_i^1) \end{array} \right] \leq \text{negl}(\lambda).$$

Theorem 2. *The above construction, TBIBKEM, has a one-of-two security in the generic group model.*

Proof. From construction we get that:

$$k_i^0 = \mathbf{r}_0^\top \begin{pmatrix} [0]_T \\ H(\text{eid}) \circ \text{pk} \end{pmatrix} \quad \text{and} \quad k_i^1 = \mathbf{r}_1^\top \begin{pmatrix} [1]_T \\ H(\text{eid}) \circ \text{pk} \end{pmatrix},$$

where $\mathbf{r}_0, \mathbf{r}_1$ are the encapsulations random values, and

$$\tilde{k}_i^0 = \text{ct}_0 \circ (\text{cm}, \pi_i^1, \sigma)^\top \quad \text{and} \quad \tilde{k}_i^1 = \text{ct}_1 \circ (\text{cm}, \pi_i^1, \sigma)^\top,$$

where:

$$\begin{aligned} \text{ct}_0 &= \mathbf{r}_0^\top \begin{pmatrix} [1]_2 & [\omega^i - \tau] & 0 \\ \text{pk} & 0 & [1]_2 \end{pmatrix} \\ \text{ct}_1 &= \mathbf{r}_1^\top \begin{pmatrix} [1]_2 & [\omega^i - \tau] & 0 \\ \text{pk} & 0 & [1]_2 \end{pmatrix} \\ \sigma &= \text{sk}(H(\text{eid}) - \text{cm}) \end{aligned}$$

Now, assume for contradiction that in the above game $\tilde{k}_i^0 = k_i^0$ and $\tilde{k}_i^1 = k_i^1$. Then putting everything together we get:

$$\begin{aligned} r_0^\top \begin{pmatrix} [0]_T \\ H(\text{eid}) \circ \text{pk} \end{pmatrix} &= r_0^\top \begin{pmatrix} [1]_2 & [\omega^i - \tau] & 0 \\ \text{pk} & 0 & [1]_2 \end{pmatrix} \circ \begin{pmatrix} \text{cm} \\ \pi_i^0 \\ \text{sk}(H(\text{eid}) - \text{cm}) \end{pmatrix} \Leftrightarrow \\ \Leftrightarrow r_0^\top \begin{pmatrix} [0]_T \\ H(\text{eid}) \circ \text{pk} \end{pmatrix} &= r_0^\top \begin{pmatrix} [1]_2 \circ \text{cm} + [\omega^i - \tau] \circ \pi_i^0 + 0 \\ \text{pk} \circ \text{cm} + 0 + [1]_2 \circ \text{sk}(H(\text{eid}) - \text{cm}) \end{pmatrix}. \end{aligned}$$

Since we are in the generic group model and $H(\text{eid}) \circ \text{pk}_2$ does not depend on the formal variable r_0 we get that

$$\begin{pmatrix} [0]_T \\ H(\text{eid}) \circ \text{pk} \end{pmatrix} = \begin{pmatrix} [1]_2 \circ \text{cm} + [\omega^i - \tau] \circ \pi_i^0 + 0 \\ \text{pk} \circ \text{cm} + 0 + [1]_2 \circ \text{sk}(H(\text{eid}) - \text{cm}) \end{pmatrix},$$

or, focusing on the first equation:

$$[1]_2 \circ \text{cm} + [\omega^i - \tau] \circ \pi_i^0 = [0]_1 \circ [1]_2.$$

Similarly, from $\tilde{k}_i^1 = k_i^1$ we get

$$[1]_2 \circ \text{cm} + [\omega^i - \tau] \circ \pi_i^1 = [1]_1 \circ [1]_2.$$

The last two equations are the KZG verification equations for $b_i = 0$ (using π_i^0 as the opening proof) and $b_i = 1$ (using π_i^1 as the opening proof) respectively. This contradicts the position-binding property of the KZG vector commitment. $\square$

4.2 Lighthouse: Our Secure Aggregation Protocol

Here, we present our main secure aggregation protocol, Lighthouse, that admits $O(1)$ server-committee communication using our threshold batched identity-based key encapsulation mechanism (TB-IB-KEM).

Setup: It happens once.

1. Sample $\text{crs}_{\text{TBIBKEM}} \leftarrow \text{TBIBKEM.Setup}(1^\lambda, 1^n)$.
2. Sample $\text{crs}_{\text{SNARK}} \leftarrow \text{SNARK.Setup}(1^\lambda, \mathcal{R}_{\text{GraphStatus}})$.
3. Distribute a random seed ν .
4. $\{\mathcal{S} \rightarrow \mathcal{D}_k\}_{k \in [L]}$: The server selects committee members $\mathcal{D}_1, \dots, \mathcal{D}_L$ at random.
5. $\{\mathcal{D}_k\}_{k \in [L]} \xrightarrow{\mathcal{S}} \{\mathcal{D}_k\}_{k \in [L]}$: The committee runs $\text{TBIBKEM.KGen}(\text{crs}_{\text{TBIBKEM}}, L, 2f)$ that outputs pk and $\text{sk}_1, \dots, \text{sk}_L$ for $i \in [L]$.⁶

Collection: It proceeds in T iterations. At each iteration $t \in [T]$ we have:

- Report:

⁶In practice this happens in a trustless manner through a server-aided distributed key generation (see [MWA⁺23]).

1. $\{\mathcal{C}_i \rightarrow \mathcal{S}\}_{i \in [n]}$: Each client i samples a fresh seed $m_{i,t} \leftarrow \{0, 1\}^\lambda$ and derives pairwise seeds by computing $h_{i,j,t} \leftarrow \text{PRF}(r_{i,j}, t)$ for all $j \in \mathcal{N}_t(i)$, where $r_{i,j}$ is a long-term Diffie-Hellman shared key. Then, clients send to the server the masked vector together with the encryptions of their masks and batched threshold encapsulations:

$$\begin{aligned} \mathbf{y}_{i,t} &:= \mathbf{x}_{i,t} + \sum_{j \in \mathcal{N}_t(i)} \pm \text{PRG}(h_{i,j,t}) + \text{PRG}(m_{i,t}), \\ \text{ct}_i^{(1)}, \hat{\text{ct}}_i^{(1)} &:= H\left(\mathbf{k}_i^{(1)}\right) \oplus m_{i,t}, \\ \left\{ \text{ct}_{i,j}^{(0)}, \hat{\text{ct}}_{i,j}^{(0)} := H\left(\mathbf{k}_{i,j}^{(0)}\right) \oplus h_{i,j,t} \right\}_{j \in \mathcal{N}_t(i)}, \end{aligned}$$

where $(\mathbf{k}_i^{(1)}, \text{ct}_i^{(1)}) \leftarrow \text{TBIBKEM}.\text{Encap}(\text{pk}, t, i \| 1)$ and $(\mathbf{k}_{i,j}^{(0)}, \text{ct}_{i,j}^{(0)}) \leftarrow \text{TBIBKEM}.\text{Encap}(\text{pk}, t, j \| 0)$, for each $j \in \mathcal{N}_t(i)$.

- Cross-check:

1. $\{\mathcal{S} \rightarrow \mathcal{D}_k\}_{k \in [L]}$: The server computes an online-offline bit vector $\mathbf{b} := (b_1, \dots, b_n)$ where $b_i = 1$ if client i sent their contribution (i.e., is online) in the report phase and $b_i = 0$ otherwise (i.e., is offline). Then, the server sends to the decryptors:

$$\begin{aligned} \text{cm} &\leftarrow \text{TBIBKEM}.\text{Digest}(\text{crs}_{\text{TBIBKEM}}, \{(\text{id}_1 \| b_1), \dots, (\text{id}_n \| b_n)\}), \\ \pi &\leftarrow \text{SNARK}.\text{Prove}\left(\text{crs}_{\text{SNARK}}, \left(\text{cm}, G_t, (1 - \delta)n, \left\lceil -\frac{\kappa}{\log \eta} \right\rceil\right); (b_1, \dots, b_n)\right), \end{aligned}$$

where G_t is the graph of the iteration and the SNARK proves that cm is a KZG vector commitment for the binary vector $\mathbf{b}$ of length n and, further, (I) at least $(1 - \delta)n$ clients are online, (II) all online clients are connected and (III) each online client has at least χ online client neighbors such that $\eta^\chi < 2^\kappa$ (see Appendix C).

2. $\{\mathcal{D}_k \rightarrow \mathcal{S}\}_{k \in [L]}$: If $\text{SNARK}.\text{Vf}\left(\text{crs}_{\text{SNARK}}, \left(\text{cm}, G_t, (1 - \delta)n, \left\lceil -\frac{\kappa}{\log \eta} \right\rceil\right)\right) = 0$, then abort. Else, send the partial decapsulation to the server:

$$\text{pd}_k \leftarrow \text{TBIBKEM}.\text{BatchPartDec}(\text{sk}_k, \text{cm}, t).$$

Reconstruction: The server reconstructs all the proper $k_i^{(1)}, k_{i,j}^{(0)}$ using the partial decryptions pd_k , and then using them recovers the proper $m_{i,t}, h_{i,j,t}$ and adds accordingly the $\mathbf{y}_{i,t}$'s to retrieve $\sum \mathbf{x}_{i,t}$. More formally, let O_t be the set of offline clients (dropouts). Then, the server

1. Computes all opening proofs $\pi_1, \dots, \pi_n$ corresponding to the KZG polynomial commitment of $b(X) = \sum_{i=1}^n b_i \mathcal{L}_i(X)$.
2. From the partial decapsulations $\{\text{pd}_k\}$ computes the full signature σ on cm .
3. For each online client $i \in [n] \setminus O_t$: Sets $\mathbf{w}_i = (\text{cm}, \pi_i, \sigma)$ and computes $\mathbf{k}_i^{(1)} := \text{ct}_i^{(1)} \circ \mathbf{w}_i$ (this is equivalent to running the $\text{TBIBKEM}.\text{Decap}$ algorithm for our construction given in Section 4.1), and retrieves $m_{i,t} := \hat{\text{ct}}_i^{(1)} \oplus H\left(\mathbf{k}_i^{(1)}\right)$.
4. For each offline client $j \in O_t$: Sets $\mathbf{w}_j = (\text{cm}, \pi_j, \sigma)$ and for each online neighbor of j , $i \in \mathcal{N}_t(j)$ computes $\mathbf{k}_{i,j}^{(0)} := \text{ct}_{i,j}^{(0)} \circ \mathbf{w}_j$ and retrieves $h_{i,j,t} := \hat{\text{ct}}_{i,j}^{(0)} \oplus H\left(\mathbf{k}_{i,j}^{(0)}\right)$.

5. Finally computes the result:

$$\sum_{i \in [n] \setminus O_t} \mathbf{x}_{i,t} := \sum_{i \in [n] \setminus O_t} \left(\mathbf{y}_{i,t} - \text{PRG}(m_{i,t}) - \sum_{j \in \mathcal{N}_t(i) \cap O_t} \pm \text{PRG}(h_{i,j,t}) \right)$$

4.3 Security Analysis

Parameter selection. We describe below the parameters relevant for the security proof.

- Threat model parameters include the maximum fraction of dropouts δ , dropout rate for decryptors (in one summation round) δ_D , maximum fraction of corruption η and corruption rate for decryptors η_D .
- The information-theoretic security parameter κ and computational security parameter λ .

We refer the reader to [MWA⁺23, Appendix C] regarding how to choose these parameters such that the inequality required in Theorem 3 is satisfied.

Security analysis. We denote the full protocol by Φ_T , which consists of a sequential execution of Setup followed by T -round execution of Collection and Reconstruction, as described in Section 4.2. To capture the security of our protocol formally, we consider our secure aggregation functionality $\mathcal{F}_{\text{Agg}}$ in Figure 2. We next show that Φ_T securely realizes $\mathcal{F}_{\text{Agg}}$ (Figure 2 without the blue lines), by formally proving the following theorem.

Theorem 3. *Let $\delta, \delta_D, \eta, \eta_D$ be the threat model parameters. Let N be the total number of clients and n be the number of clients for summation in each round. Assuming the existence of a PKI, a trusted source of initial randomness, a hash function H and a PRG (both modeled as a random oracle), a secure PRF, a succinct non-interactive argument of knowledge (SNARK) and a secure threshold batched identity-based key encapsulation mechanism (TB-IB-KEM), if $2\delta_D + \eta_D < 1/3$, then the protocol Φ_T securely realizes $\mathcal{F}_{\text{Agg}}$ in the random oracle model and in the presence of a malicious adversary with static corruptions controlling η fraction of N clients (and the server), except with probability at most $Tn \cdot 2^{-\kappa+1} + \text{negl}(\lambda)$.*

Proof. We remark that the assumption regarding the trusted source of initial randomness is needed only for the Setup protocol, which is (nearly) identical to the Setup protocol of [MWA⁺23] (with the addition of SNARK common reference string generation), and hence, the security of this protocol follows from [MWA⁺23, Theorem 1].

Next, we present the proof for a single round of collection, that is we prove that the Collection protocol (from Section 4) for round t securely realizes the ideal functionality $\mathcal{F}_{\text{Sum}}^{(t)}$ (Figure 3) in the random oracle model. Our proof follows a similar structure to that of [MWA⁺23, Theorem 3]. Observe that in the functionality $\mathcal{F}_{\text{Sum}}^{(t)}$ the output sum is not determined by the actual dropout set O_t , but instead M_t sent by the adversary. For the sake of simplicity, we omit the round number t when there is no ambiguity. We assume that the adversary $\mathcal{A}$ controls a set of clients in $[N]$ with the constraint $2\delta_D + \eta_D < 1/3$. Let C be the set of corrupted clients and D the set of decryptors. Hence, we have that the corrupted decryptors form a set $C \cap D$ and $|C \cap D| < L/3$, where $L = |D|$, and from the analysis given in [MWA⁺23, Appendix A] we have $|C|/|S_t| \leq \eta$. We consider two distinct cases for the security proof, that is when the server is corrupted and when it is not.

Parties: clients in S_t and a server.

Parameters: dropout rate δ and corruption rate η over n_t clients.

- $\mathcal{F}_{\text{Sum}}^{(t)}$ receives a set O_t such that $|O_t|/|S_t| \leq \delta$, and from the adversary $\mathcal{A}$ a set of corrupted parties C , and $\mathbf{x}_{i,t}$ from client $i \in S_t \setminus (O_t \cup C)$.
- $\mathcal{F}_{\text{Sum}}^{(t)}$ sends S_t and O_t to $\mathcal{A}$, and asks $\mathcal{A}$ for a set M_t : if $\mathcal{A}$ replies with $M_t \subseteq S_t \setminus O_t$ such that $|M_t|/|S_t| \geq 1 - \delta$, then $\mathcal{F}_{\text{Sum}}^{(t)}$ computes $\mathbf{y}_t = \sum_{i \in M_t \setminus C} \mathbf{x}_{i,t}$, and continues; otherwise $\mathcal{F}_{\text{Sum}}^{(t)}$ sends abort to all the honest parties.
- Depending on whether the server is corrupted by $\mathcal{A}$:
 - If the server is corrupted by $\mathcal{A}$, then $\mathcal{F}_{\text{Sum}}^{(t)}$ outputs $\mathbf{y}_t$ to all the parties corrupted by $\mathcal{A}$.
 - If the server is not corrupted by $\mathcal{A}$, then $\mathcal{F}_{\text{Sum}}^{(t)}$ asks $\mathcal{A}$ for a shift $\mathbf{a}_t$ and outputs $\mathbf{y}_t + \mathbf{a}_t$ to the server.

Figure 3: Functionality $\mathcal{F}_{\text{Sum}}^{(t)}$ [MWA⁺23]

Case 1. We start with the case where the server is corrupted by the adversary $\mathcal{A}$. First, we construct a simulator $\mathcal{S}$ in the ideal world where it runs $\mathcal{A}$ as a subroutine. We consider that the dropout set is not provided ahead of the protocol run but instead provided during the execution (as also done in [BIK⁺17, MWA⁺23]). For this we assume (analogous to the security proof of [MWA⁺23, Theorem 3]) that both the simulator $\mathcal{S}$ and ideal functionality $\mathcal{F}_{\text{Sum}}^{(t)}$ have access to an oracle $\mathcal{O}_{\text{drop}}$ that provides the dropout set O_t . The simulator $\mathcal{S}$ for round t is defined as follows:

1. $\mathcal{S}$ receives a set O_t from the oracle $\mathcal{O}_{\text{drop}}$ and sends O_t to $\mathcal{F}_{\text{Sum}}^{(t)}$.
2. $\mathcal{S}$ receives the sets M_t and C from the adversary $\mathcal{A}$, and obtains $\mathbf{y}_t$ from $\mathcal{F}_{\text{Sum}}^{(t)}$.
3. (Setup phase) $\mathcal{S}$ simulates the honest decryptors $k \in D \setminus C$, and thus, generates $\{\text{sk}_k\}_{k \in D \setminus C}$ and the corresponding the public key pk , the common reference strings $\text{crs}_{\text{TBIBKEM}}$, $\text{crs}_{\text{SNARK}}$ and obtains the distributed random seed ν .
4. (Report phase) $\mathcal{S}$ simulates the honest clients $i \in M_t \setminus C$ to the adversary $\mathcal{A}$ as follows:
 - Obtain the pairwise secrets $r_{i,j}$ by querying the PKI, and compute the pairwise seeds $h_{i,j} \leftarrow \text{PRF}(r_{i,j}, t)$.
 - Sample the individual seed $m_i \leftarrow \{0, 1\}^\lambda$.
 - Sample a random vector $\mathbf{y}'_i \leftarrow \{0, 1\}^\ell$, and a random input $\mathbf{x}'_i \in \{0, 1\}^\ell$ subject to the constraint that $\sum_{i \in M_t \setminus C} \mathbf{x}'_i = \mathbf{y}_t$, and program the random oracle PRG (i.e., we assume a perfect PRG) as

$$\text{PRG}(m_i) = \mathbf{y}'_i - \mathbf{x}'_i - \sum_{j \in \mathcal{N}_t(i)} \pm \text{PRG}(h_{i,j}).$$

- Let $\mathcal{K} := \{\mathbf{k}_i^{(1)} : i \in S_t \setminus C\} \cup \{\mathbf{k}_{j,i}^{(0)} : i, j \in M_t \setminus C \wedge j \in \mathcal{N}(i)\}$. If $\mathcal{A}$ queries $H(\mathbf{k})$ for some $\mathbf{k} \in \mathcal{K}$, then abort with the message `AbortEarly`.

- Let $\mathcal{E} := \{\widehat{\text{ct}}_i^{(1)} : i \in S_t \setminus C\} \cup \{\widehat{\text{ct}}_{j,i}^{(0)} : i, j \in M_t \setminus C \wedge j \in \mathcal{N}(i)\}$. Replace each $\widehat{\text{ct}} \in \mathcal{E}$ with a uniformly random bitstring $\sigma \leftarrow \{0, 1\}^\lambda$.
 - Compute the ciphertexts $(k_i^{(1)}, \text{ct}_i^{(1)}) \leftarrow \text{TBIBKEM.Encap}(\text{pk}, t, i \| 1)$ and $\{(k_{i,j}^{(0)}, \text{ct}_{i,j}^{(0)}) \leftarrow \text{TBIBKEM.Encap}(\text{pk}, t, j \| 0)\}_{j \in \mathcal{N}(i)}$.
 - Send $y'_i, \text{ct}_i^{(1)}, \widehat{\text{ct}}_i^{(1)}, \{\text{ct}_{i,j}^{(0)}\}_{j \in \mathcal{N}(i)}$ and $\{\widehat{\text{ct}}_{i,j}^{(0)}\}_{j \in \mathcal{N}(i)}$ to the server.
5. (Cross-check phase) Upon receiving (cm, π) from the server, $\mathcal{S}$ simulates the honest decryptors $k \in D \setminus C$ to the adversary $\mathcal{A}$ as follows:
- If $\text{SNARK.Vf}(\text{crs}_{\text{SNARK}}, (\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil)) = 0$, then $\mathcal{S}$ sends abort to $\mathcal{F}_{\text{Sum}}^{(t)}$, outputs whatever $\mathcal{A}$ outputs and halts.
 - Else, run $\mathbf{b} := (b_1, \dots, b_n) \leftarrow \text{SNARK.Ext}_{\mathcal{A}}(\text{crs}_{\text{SNARK}}, (\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \pi)$, if the knowledge soundness extractor fails or $((\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \mathbf{b}) \notin \mathcal{R}_{\text{GraphStatus}}$, then abort with the message AbortSNARK.
 - Program the random oracle H , for $i \in S_t \setminus C$, as follows:
 - if $b_i = 1$, then program $H(k_i^{(1)}) := \sigma_i^{(1)} \oplus m_i$;
 - if $b_i = 0$, then program $H(k_{j,i}^{(0)}) := \sigma_{j,i}^{(0)} \oplus h_{j,i}$ for all $j \in \mathcal{N}(i)$ such that $j \in M_t \setminus C$,
 where $\sigma_i^{(1)}$ and $\sigma_{j,i}^{(0)}$ are the corresponding random bitstrings used during the report phase.
 - Otherwise, compute $\text{pd}_k \leftarrow \text{TBIBKEM.BatchPartDec}(\text{sk}_k, \text{cm}, t)$ and send pd_k to the server.
 - If $\exists i \in M_t \setminus C$ such that $k_i^{(1)} = \widetilde{k}_i^{(1)}$ and $k_{j,i}^{(0)} = \widetilde{k}_{j,i}^{(0)}$, for some $j \in M_t \setminus C$ and $j \in \mathcal{N}(i)$, then abort with the message AbortBinding, where $k_i^{(1)} \leftarrow \text{TBIBKEM.Decap}(\text{crs}, t, (\text{id}_1 \| b_1, \dots, \text{id}_n \| b_n), \{\text{pd}_k\}_{k \in D \setminus C}, \text{id} \| 1, \text{ct}_i^{(1)})$ and $k_{j,i}^{(0)} \leftarrow \text{TBIBKEM.Decap}(\text{crs}, t, (\text{id}_1 \| b_1, \dots, \text{id}_n \| b_n), \{\text{pd}_k\}_{k \in D \setminus C}, \text{id} \| 0, \text{ct}_{j,i}^{(0)})$.
 - If $\mathcal{A}$ queries the random oracle $H(k_{*,i}^{(\overline{b}_i)})$ for some $i \in M_t \setminus C$ (i.e., if $b_i = 0$, then $\mathcal{A}$ queries the random oracle on $h_i^{(1)}$, and if $b_i = 1$, then it queries on $h_{j,i}^{(0)}$ for some $j \in \mathcal{N}(i)$ and $j \in M_t \setminus C$), then abort with the message AbortEnc.

Next, we construct a series of hybrids, starting from the real world to the ideal world execution. Hereafter, let $H_i \approx H_{i+1}$ denote $|\Pr[H_i = 1] - \Pr[H_{i+1} = 1]| \leq \text{negl}(\lambda)$.

Hybrid H_0 : This corresponds to the real protocol (Section 4.2).

Hybrid H_1 : This is identical to H_0 except that the simulator $\mathcal{S}$ invokes the knowledge-soundness extractor $\mathbf{b} \leftarrow \text{SNARK.Ext}_{\mathcal{A}}(\text{crs}_{\text{SNARK}}, (\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \pi)$, and it aborts with the message AbortSNARK if the extraction fails or $((\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \mathbf{b}) \notin \mathcal{R}_{\text{GraphStatus}}$.

Notice that for a proof π that successfully verifies, the extraction failing or extracted witness $\mathbf{b}$ satisfying $((\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \mathbf{b}) \notin \mathcal{R}_{\text{GraphStatus}}$ correspond to the winning conditions of knowledge-soundness game (Definition 1). Hence, by the knowledge-soundness of SNARK we have that this event occurs with negligible probability, and it follows that $H_0 \approx H_1$.

Hybrid H₂: This is identical to H₁ except that the simulator $\mathcal{S}$ aborts with the message AbortBinding if after computing the partial decryptions $\{\text{pk}_k\}_{k \in D \setminus C}$, it holds that there exists some $i \in M_t \setminus C$, such that $\mathbf{k}_i^{(1)} = \widehat{\mathbf{k}}_i^{(1)}$ and $\mathbf{k}_{j,i}^{(0)} = \widehat{\mathbf{k}}_{j,i}^{(0)}$, for some $j \in M_t \setminus C$ and $j \in \mathcal{N}(i)$, where for $\widehat{\mathbf{k}}_i^{(1)} \leftarrow \text{TBIBKEM.Decap}(\text{crs}, t, (\text{id}_1 \| b_1, \dots, \text{id}_n \| b_n), \{\text{pd}_k\}_{k \in D \setminus C}, \text{id} \| 1, \text{ct}_i^{(1)})$ and $\widehat{\mathbf{k}}_{j,i}^{(0)} \leftarrow \text{TBIBKEM.Decap}(\text{crs}, t, (\text{id}_1 \| b_1, \dots, \text{id}_n \| b_n), \{\text{pd}_k\}_{k \in D \setminus C}, \text{id} \| 0, \text{ct}_{j,i}^{(0)})$.

Clearly, the hybrids H₁ and H₂ are identical as long as the AbortBinding event does not happen. We can bound the probability of the event AbortBinding happening by a reduction to the one-of-two security of TBIBKEM (Definition 3). Concretely, the reduction receives $(\text{crs}, \text{pk}, \{\text{sk}_i\}_{i \in [L]}, \{\text{ct}_i^{(b)}\}_{b \in \{0,1\}})$ from the one-of-two challenger of TBIBKEM, and it computes the ciphertexts as in the previous hybrid except for the i -th client uses $\text{ct}_i^{(b)}$. Upon receiving some (cm, π) from $\mathcal{A}$, the reduction extracts $\mathbf{b} \leftarrow \text{SNARK.Ext}_{\mathcal{A}}(\text{crs}_{\text{SNARK}}, (\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \pi)$, parses $\mathbf{b} := (b_1, \dots, b_n)$ and forwards $(\text{cm}, \{\text{id}_1 \| b_1, \dots, \text{id}_n \| b_n\})$ to the one-of-two challenger.

We note that the reduction simulates the view to the adversary perfectly. Moreover, it is easy to see that the probability of event AbortEnc happening can be upper bounded by $\text{Adv}_{\text{TBIBKEM}}^{\text{one-of-two}}$, which is negligible by assumption. Hence, we have that $H_1 \approx H_2$.

Hybrid H₃: Let the set $\mathcal{K} := \{\mathbf{k}_i^{(1)} : i \in S_t \setminus C\} \cup \{\mathbf{k}_{j,i}^{(0)} : i, j \in M_t \setminus C \wedge j \in \mathcal{N}(i)\}$. This hybrid is identical to H₂ except that the simulator $\mathcal{S}$ aborts with the message AbortEarly if the adversary makes a random oracle query $H(\mathbf{k})$ for some $\mathbf{k} \in \mathcal{K}$ before it receives the ciphertexts from the honest clients (at the end of the report phase).

Clearly, the hybrids H₂ and H₃ are identical as long as the AbortEarly event does not happen. Therefore, we need to bound the probability $\Pr[\text{AbortEarly}]$. Let q_H denote the total number of queries that $\mathcal{A}$ makes to the random oracle H and q_i denote the i -th random oracle query. It holds that

$$\begin{aligned} \Pr[\text{AbortEarly}] &\leq \Pr[q_1 \in \mathcal{K}] + \sum_{i=2}^{q_H} \Pr \left[q_i \in \mathcal{K} \mid \bigwedge_{j=1}^{i-1} q_j \notin \mathcal{K} \right] \\ &= \sum_{i=1}^{q_H} \frac{|\mathcal{K}|}{2^\lambda - (i-1)} \leq \sum_{i=1}^{q_H} \frac{|\mathcal{K}|}{2^\lambda - q_H} \leq \frac{|\mathcal{K}| \cdot q_H}{2^{\lambda-1}} \leq \text{negl}(\lambda). \end{aligned}$$

Above holds because TBIBKEM.Encap samples a key $\mathbf{k}$ uniformly from the key space. Moreover, the last inequality follows from the fact that the adversary is a PPT algorithm, and hence, both $\mathcal{K}$ and q_H are polynomially bounded. Therefore, we have that $H_2 \approx H_3$.

Hybrid H₄: Let $\mathcal{K} := \{\mathbf{k}_i^{(1)} : i \in S_t \setminus C\} \cup \{\mathbf{k}_{j,i}^{(0)} : i, j \in M_t \setminus C \wedge j \in \mathcal{N}(i)\}$ and $\mathcal{E} := \{\widehat{\text{ct}}_i^{(1)} : i \in S_t \setminus C\} \cup \{\widehat{\text{ct}}_{j,i}^{(0)} : i, j \in M_t \setminus C \wedge j \in \mathcal{N}(i)\}$. This hybrid is identical to H₃ except that the simulator $\mathcal{S}$ does not program $H(\mathbf{k})$ for $\mathbf{k} \in \mathcal{K}$ during the report phase, and instead during the report phase it replaces each $\widehat{\text{ct}} \in \mathcal{E}$ with a uniformly random bitstring $\sigma \leftarrow \{0, 1\}^\lambda$. Moreover, upon receiving (cm, π) from $\mathcal{A}$, the simulator $\mathcal{S}$ extracts $\mathbf{b} \leftarrow \text{SNARK.Ext}_{\mathcal{A}}(\text{crs}_{\text{SNARK}}, (\text{cm}, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \pi)$, parses $\mathbf{b} := (b_1, \dots, b_n)$ and programs the random oracle for $i \in S_t \setminus C$ as follows:

- if $b_i = 1$, then program $H(\mathbf{k}_i^{(1)}) := \sigma_i^{(1)} \oplus m_i$;
- if $b_i = 0$, then program $H(\mathbf{k}_{j,i}^{(0)}) := \sigma_{j,i}^{(0)} \oplus h_{j,i}$ for all $j \in \mathcal{N}(i)$ such that $j \in M_t \setminus C$,

where $\sigma_i^{(1)}$ and $\sigma_{j,i}^{(0)}$ are the corresponding random bitstrings used during the report phase.

Since we know that AbortEarly event happens with negligible probability due to the hybrid H_3 , it follows that the keys $k \in \mathcal{K}$ are still uniformly random after $\mathcal{A}$ receives the ciphertexts at the end of the report phase. Hence, we have that the ciphertexts $\hat{ct} \in \mathcal{E}$ are uniform as well. Therefore, replacing the ciphertexts $\hat{ct} \in \mathcal{E}$ with uniformly random bitstrings $\sigma \leftarrow \{0, 1\}^\lambda$ do not change their distributions. Additionally, even though $\mathcal{S}$ does not program $H(k)$ for $k \in \mathcal{K}$ during the report phase, it keeps the random oracle consistent by programming it during the cross-check phase according to the online-offline vector b chosen by $\mathcal{A}$. Hence, it follows that $H_3 \approx H_4$.

Hybrid H_5 : Let $\mathcal{E}' := \{ct_i^{(1)} : i \in M_t \setminus C\} \cup \{ct_{j,i}^{(0)} : i, j \in M_t \setminus C \wedge j \in \mathcal{N}(i)\}$. This hybrid is identical to H_4 except that the simulator $\mathcal{S}$ aborts with the message AbortEnc if after the cross-check step the adversary $\mathcal{A}$ queries the random oracle $H(k_{*,i}^{(\overline{b_i})})$ for some $i \in M_t \setminus C$ (i.e., if $b_i = 0$, then $\mathcal{A}$ queries the random oracle on $h_i^{(1)}$, and if $b_i = 1$, then it queries on $h_{j,i}^{(0)}$ for some $j \in \mathcal{N}(i)$ and $j \in M_t \setminus C$).

Clearly, the hybrids H_4 and H_5 are identical as long as the AbortEnc event does not happen. We can bound the probability of the event AbortEnc happening by a reduction to the OW-PCA security of TBIBKEM (Definition 2). Concretely, the reduction receives $(crs, pk, \{sk_k\}_{k \in C})$ from the OW-PCA challenger of TBIBKEM, then it guesses some $i' \in M_t \setminus C$ and $b' \in \{0, 1\}$ and sends the pair $(t, id_{i'} \| b')$ to the OW-PCA challenger. It receives back some ct' from the OW-PCA challenger and it replaces the ciphertext $ct_{*,i'}^{b'} \in \mathcal{E}$ with ct' (i.e., it sets $ct_{i'}^{(b')} := ct'$ if $b' = 1$, and $ct_{j,i'}^{(b')} := ct'$ for some $j \in M_t \setminus C$ otherwise). The reduction computes the rest of the ciphertexts as in the previous hybrid.

We note that the reduction does not need to guess when the AbortEnc event happens, as it has access to the $\mathcal{O}_{\text{Check}}$ oracle from the OW-PCA challenger. More precisely, upon adversary $\mathcal{A}$ querying the random oracle on the challenge key, i.e., $H(k')$, the reduction can invoke $\mathcal{O}_{\text{Check}}(k', ct', \{id_1, \dots, id_n\}, id_{i'} \| b', t)$ (where we have that $id_{i'} \| b' \in \{id_1, \dots, id_n\}$). If the oracle returns 1, then the reduction also outputs the queried k' .

If $\mathcal{A}$ returns some (cm, π) , such that $b \leftarrow \text{SNARK.Ext}_{\mathcal{A}}(crs_{\text{SNARK}}, (cm, G_t, (1 - \delta)n, \lceil -\frac{\kappa}{\log \eta} \rceil), \pi)$, and for $b := (b_1, \dots, b_n)$ it holds that $id_{i'} \| b' \in \{id_1 \| b_1, \dots, id_n \| b_n\}$, then clearly the reduction guessed wrong and it aborts. Otherwise, the reduction invokes the oracle $\mathcal{O}_{\text{PartDec}}(\{id_1 \| b_1, \dots, id_n \| b_n\}, t)$ to obtain the partial decryptions $\{pd_k\}_{k \in D \setminus C}$, which it returns to $\mathcal{A}$.

We note that the reduction simulates the view to the adversary perfectly, and additionally, the adversary $\mathcal{A}$ cannot decrypt all the ciphertexts from $\mathcal{E}'$ using the partial decryptions due to the one-of-two security of TBIBKEM (as argued in hybrid H_2). Therefore, we have that the probability of the event AbortEnc happening can be upper bounded by $|\mathcal{E}'| \cdot \text{Adv}_{\text{TBIBKEM}}^{\text{ow-pca}}$, which is negligible by the security of TBIBKEM and the fact that $|\mathcal{E}'|$ is polynomially bounded. Hence, we have that $H_4 \approx H_5$.

Hybrid H_6 : This hybrid is identical to H_5 except that the simulator $\mathcal{S}$ upon sampling the random individual masks $m'_i \leftarrow \{0, 1\}^\lambda$ (and before computing any ciphertexts), programs the random oracle PRG (i.e., we assume that we have a perfect PRG). Concretely, for all $i \in M_t \setminus C$, $\mathcal{S}$ samples a random vector $y'_i \leftarrow \{0, 1\}^\ell$ and programs $\text{PRG}(m'_i)$ as

$$\text{PRG}(m'_i) = y'_i - x_i - \sum_{j \in \mathcal{N}_t(i)} \pm \text{PRG}(h_{i,j}).$$

We note that the view of the adversary $\mathcal{A}$ regarding y_i in this hybrid is statistically indistinguishable to that in the previous hybrid. Moreover, given that a PPT adversary $\mathcal{A}$ makes at most $q_{\text{PRG}} = \text{poly}(\lambda)$ queries to the random oracle PRG we have that $\mathcal{A}$ notices this programming with negligible probability (following an analogous argument as in hybrid H_3). Hence, it follows that $H_5 \approx H_6$.

Hybrid H₇: This hybrid is identical to H₆ except that $\mathcal{S}$ programs the random oracle PRG as

$$\text{PRG}(m'_i) = \mathbf{y}'_i - \mathbf{x}'_i - \sum_{j \in \mathcal{N}_t(i)} \pm \text{PRG}(h_{i,j}),$$

where $\mathbf{x}'_i \in \{0, 1\}^\ell$ is chosen randomly subject to the constraint $\sum_{i \in M_t \setminus C} \mathbf{x}_i = \sum_{i \in M_t \setminus C} \mathbf{x}'_i$. The advantage of the adversary in this hybrid is equal to the probability that the graph becomes disconnected or that there is an isolated node in $(S_t \setminus O_t) \setminus C$, as in these cases the server learns the input vector $\mathbf{x}_i$ in the clear. However, by [BIK⁺17, Lemma 6.1] the probability of this is bounded by $2^{-\kappa}$. Moreover, if $\mathcal{A}$ cheats by using some M_t where the graph formed by nodes in M_t is not connected, then due to the knowledge-soundness of π the proof does not verify, and $\mathcal{S}$ simulates honest decryptors aborting. Therefore, it follows that $H_6 \approx H_7$.

We note that this last hybrid is exactly the ideal world execution. Next, we analyze the success probability of the simulation. Let κ_1 be the security parameter for graph connectivity from [MWA⁺23, Lemma 1], and κ_2 denote the security parameter for the check that the client has at least χ online neighbors. The simulation can fail if either (i) the graph gets disconnected, or (ii) there exists a client in S_t such that all of its neighbors are malicious. We have that the former happens with probability $2^{-\kappa_1}$, and the later is bounded by $n \cdot 2^{-\kappa_2}$. Following the combinatorial arguments from the proof of [MWA⁺23, Theorem 3], the probability that opposite of event (ii) happens is $(1 - \eta^\chi)^n \approx 1 - n\eta^\chi$ (assuming η^χ is very small). Since $n\eta^\chi \leq n \cdot 2^{-\kappa_2}$, we have that the overall failure probability of the simulation is $2^{-\kappa_1} + n \cdot 2^{-\kappa_2} + \text{negl}(\lambda)$, where $\text{negl}(\lambda)$ comes from the computational indistinguishability advantage in the above hybrids.

Case 2. The simulation of the case where server is not corrupted by the adversary $\mathcal{A}$ is identical to the previous case, except that the simulator $\mathcal{S}$ only needs to simulate the honest clients and decryptors. We also remark that when simulating the honest server we can compute the proof π honestly since the whole graph and online-offline bit vector can be publicly derived from the distributed random seed ν , and hence, we do not require any zero-knowledge property from the underlying SNARK.

This completes the proof for any single round $t \in [T]$, i.e., the protocol given in Section 4.2 for round t securely realizes $\mathcal{F}_{\text{Sum}}^{(t)}$ when $\delta_D + \eta_D < 1/3$, except with probability $2^{-\kappa_1} + n \cdot 2^{-\kappa_2} + \text{negl}(\lambda) \leq n \cdot 2^{-\kappa+1} + \text{negl}(\lambda)$, where $\kappa = \min\{\kappa_1, \kappa_2\}$.

Multi-round security follows by an analogous argument as in the proof of [MWA⁺23, Theorem 3]. Concretely, there are two additional steps to consider on top of the single-round proof:

1. The set S_t is generated from PRG (see the ChooseSet algorithm from [MWA⁺23, Figure 1]), and hence, we have to program the random oracle PRG such that the ChooseSet algorithm outputs S_t .
2. We need to perform a reduction to the PRF security for the per-round pairwise masks $h_{i,j,t}$ computed from $\text{PRF}(r_{i,j}, t)$.

Finally, we can bound the failure probability for T rounds as follows: since the failure probability for a single round is $p = n \cdot 2^{-\kappa+1} + \text{negl}(\lambda)$, the probability that for all T rounds the protocol is secure is therefore $1 - (1 - p)^T$, which approximates to $1 - T \cdot p$ when $T \cdot p \ll 1$. Hence, the failure probability for T protocol is $Tn \cdot 2^{-\kappa+1} + \text{negl}(\lambda)$. This concludes the proof of Theorem 3. $\square$

On the necessity of idealized models. Prior lower bounds [BCGL⁺25] demonstrate that simulation-secure, one-shot aggregation against a corrupted server with sublinear communication is impossible in the standard model. Consequently, reliance on idealized models like the Random Oracle Model (ROM) seems to be unavoidable. However, we note that, unlike prior works, our protocol additionally relies on the

Generic Group Model (GGM). Importantly, our core protocol makes only black-box use of its underlying primitives and is proven secure in the ROM. Our reliance on the GGM is limited to our concrete instantiations (the underlying SNARK and our TBIBKEM). As the protocol is modular, this GGM dependency could be eliminated as primitives evolve. For instance, by using RO-model SNARKs [LPS25] or standard-model TBIBKEM constructions (potentially building on [GWW25])

4.4 Robustness of Pairwise Masking Against Input Manipulation

Recent analyses of single-server secure aggregation protocols [MLLP25] have highlighted a specific “Man-in-the-Middle” vulnerability in schemes that lack *input soundness*—specifically, protocols where the client’s vector payload is not cryptographically bound to their identity via a signature (e.g., Willow [BCGL⁺25]). In this attack, a malicious server replaces the ciphertexts of honest clients with encryptions of zero (or arbitrary values) to mathematically isolate and expose a target client’s input in the final aggregate. While Lighthouse similarly does not require signatures over the high-dimensional vector payload, we argue that its privacy guarantees remain intact against this attack due to its “double-masking” architecture and conditional committee logic.

Unlike schemes relying solely on homomorphic encryption [MLLP25, BCGL⁺25], pairwise masking-based systems such as ours and Flamingo [MWA⁺23] employ a dual layer of protection consisting of individual masks ($m_{i,t}$) and pairwise masks ($h_{i,j,t}$). This structure creates a “deadlock” that prevents a server from unmasking a specific target, even if the server possesses the ability to tamper with vector payloads.

Consider a scenario where a malicious server attempts to isolate a target client $\mathcal{C}_i$ by replacing the vectors of all neighbors $j \in \mathcal{N}(\mathcal{C}_i)$ with zero vectors. To recover $\mathcal{C}_i$ ’s input $\mathbf{x}_i$, the server must remove both the individual mask and the aggregated pairwise masks:

$$\text{mask}_i = \text{PRG}(m_{i,t}) + \sum_{j \in \mathcal{N}(i)} \pm \text{PRG}(h_{i,j,t})$$

Flamingo’s committee enforces a strict mutual exclusivity rule for decryption in any given round: a client is labeled either “online” or “offline”.

1. If i is labeled “online”, the committee decrypts only the individual mask seed $m_{i,t}$. The pairwise masks $h_{i,j,t}$ remain encrypted and unknown to the server.
2. To remove the remaining pairwise masks $\text{PRG}(h_{i,j,t})$, the server must acquire the pairwise keys. However, the committee releases these keys *only* if the corresponding neighbor j is labeled Offline.

This creates a dilemma for the adversary. To unmask i , the server must label i as “online” (to get $m_{i,t}$) but essentially label all (honest) neighbors as “offline” (to get the pairwise keys $h_{i,j,t}$). However, the committee performs a robust threshold check, aborting the protocol if the number of “online” clients falls below a critical threshold $(1 - \delta)n_t$. Consequently, the server cannot label a sufficient number of neighbors as “offline” to retrieve the necessary keys without triggering an abort. The target’s input thus remains information-theoretically hidden by the unremoved pairwise masks of the “online” neighbors.

Moreover, the individual mask $m_{i,t}$ functions as a cryptographic pad. While the lack of binding allows the server to substitute the ciphertext, this substitution provides no advantage in decrypting the original input, as the keys required to unmask the data remain protected by the committee’s threshold logic as described.

4.5 Robustness Against Malicious Labeling in and Across Iterations

A critical attack vector in server-mediated secure aggregation is *view equivocation* or malicious labeling, where a corrupt server selectively misrepresents the status of clients (i.e., labeling them online or offline)

to extract masking key materials. Such manipulation can potentially trick the decryptors committee into releasing both types of masks for some client. We analyze how our protocol mitigates this threat thanks to our underlying encryption scheme TB-IBE.

Defense against equivocation. In a standard malicious labeling attack [MWA⁺23], the server attempts to present inconsistent views of the client set $\mathbf{b}$ to different (honest) committee members to harvest sufficient partial decryption shares for both pairwise masks and individual mask with the help of a quorum of malicious committee members. Our protocol neutralizes this through the underlying threshold Batched Identity-Based Encryption scheme requiring a quorum of $2f + 1$ committee members sign the same digest (i.e., commitment cm) for correct decryption (see Definition 2). Since the server must aggregate valid partial decryptions from a supermajority to reconstruct the mask, it cannot successfully equivocate and must present a single, consistent view to the quorum. If the server attempts to modify the client status vector for a subset of honest decryptors, it will fail to accumulate the sufficient threshold of consistent partial signatures. This is because it would need $f + 1$ partial signatures from honest decryptors requiring least an honest decryptor sign two different digests, which is not possible. Note that here we also leverage the binding property of the KZG commitment scheme, where the probability of finding two distinct vectors $\mathbf{b} \neq \mathbf{b}'$ such that $\text{cm} = \text{cm}'$ is negligible. Hence, a valid signature on the digest uniquely binds the server to a specific client status vector.

Defense against replay attack. A more subtle variation involves *cross-iteration* attacks, where a server attempts to “time-travel” by combining ciphertexts from different iterations. For instance, a server might harvest pairwise keys $\{h_{i,j,t}\}$ for an offline client C_i in iteration t and attempt to obtain its individual masks $m_{i,t}$ at round t' (instead of $m_{i,t'}$) by nevertheless labeling them online. Our protocol enforces strict *temporal isolation* by cryptographically binding all ciphertexts to unique session/iteration identifier (i.e., batch label eid). This binding ensures that partial decryption keys released for one iteration are mathematically incompatible to use for decrypting ciphertexts from any other iteration. Consequently, even if a server accumulates a history of ciphertexts from previous iterations, it yields no advantage in mounting a replay attack. Note that Flamingo [MWA⁺23] explicitly requires signatures on the client’s ciphertexts and incorporating the iteration number in cross-checking agreement by the decryptors.

5 Secure Aggregation with Dynamic Participation

In our main protocol (Section 4.1), the client universe $[N]$ is statically defined prior to the start of the training. This imposes a limitation in dynamic, real-world deployments such as the Web, where client devices (e.g., browsers) are typically ephemeral and unknown to the server [BCGL⁺25]. Ideally, the aggregation mechanism should support an *open-world* model: a client should be able to contribute without having prior knowledge of or establishing cryptographic dependencies with the identities of other participating clients. Therefore, we need to devise a protocol extension that the clients participating in a given aggregation iteration are not determined up front, but decided/chosen dynamically as the protocol proceeds. This enables dynamic client participation, where any new client can join at any time and contribute without prior coordination with other clients or a per-iteration registration phase. Observe that the root cause of fixing the set of clients in previous pairwise masking-based solutions such as Bell et al. [BBG⁺20] and Flamingo [MWA⁺23] is that each client needs to know all the other clients as their potential neighbors, even at the setup phase [MWA⁺23].

Our idea to support dynamic participation is to have (new) clients establish their communication graph only with the set of committee members (i.e., decryptors) who are already known, making them their neighbors. Moreover, instead of using two types of masks (i.e., pairwise and individual) each client now



Figure 4: **Topology Comparison.** (a) In the standard mesh (Flamingo), a new client must register and establish pairwise keys with random peer neighbors to ensure mask cancellation. (b) In our dynamic extension, clients connect solely to the stable committee. A new client can join spontaneously by deriving zero-sum masks with the committee public keys, eliminating the need for peer-to-peer coordination.

only uses a single type of mask to blind their input vector with a summation of all pairwise masks with the committee members, creating a *zero-sum* random masking. These two changes thus eliminate the need for fixing the set of clients and allow for any number of (new) clients to join at an iteration. Given that we no longer rely on double masking similar to our main protocol and there is no graph dependency with neighbors for mask cancellation, client drop-out is no longer an issue. We remark that the feasibility of zero-sum masking is directly dictated by the stability and overlap of the neighbor set (i.e. committee). In a random mesh topology, neighbors are disjoint and unreliable, meaning a zero-sum is statistically impossible to maintain without a fallback layer. However, we need to handle the decryptor drop-out as the contribution from every single decryptor-summation of all pairwise masks with the online clients-is needed to derive the final result. This is where Threshold Batched Identity-Based Encryption (TB-IBE) comes into play that allows to batch decrypt all the masking seeds corresponding to unresponsive decryptors and clients. In Figure 4, we illustrate the topology shift from a static peer-to-peer (p2p) mesh⁷ to a dynamic client-to-committee bipartite structure.

We remark that in principle we do not need to deploy PKI assumption for the clients, but merely for the small set of committee members.⁸ However, we present the protocol assuming the existence of PKI to allow for efficient Diffie-Hellman key exchanges between the (new) client and the committee members. While our primary contribution remains the optimal $O(1)$ efficiency of the static baseline protocol, supporting new clients requires committee members to perform only two highly parallelizable local computational steps: establishing shared seeds via standard public-key operations and expanding the symmetric PRG masks. Furthermore, following their ephemeral participation in the current iteration, these dynamic clients can either leave the system or gracefully transition into the static client pool for subsequent iterations by asynchronously executing the standard peer-to-peer setup phase.

5.1 Construction

In this section, we present a protocol with dynamic client participation that admits $O(\ell)$ communication and linear computation cost in the number of newly joined clients for committee members. Importantly, this protocol has each decryptor also contribute towards the aggregation, allowing to piggyback on the $O(\ell)$ communication cost for input vector. As we discussed in Section 3, we consider this as an extension

⁷Note that using p2p here is merely for visualization, as all communication is routed through the server.

⁸Each client can simply send a ciphertext encrypting a random value under the public key of each decryptor to enable shared key derivation.

to our main protocol where new clients can run to be able to contribute to a given aggregation iteration without re-running the setup phase, enabling hybrid client participation.

- **Setup** : The protocol is the same as in Section 4.2.
- **Collection** : It proceeds in T iterations. At each iteration $t \in [T]$ we have:

– **Report-Client**:

1. $\{\mathcal{C}_i \rightarrow \mathcal{S}\}_{i \in [n]}$: Each new client i derives pairwise seeds by computing

$$h_{i,j,t} := \text{PRG}(r_{i,j}, t) \quad \text{for all } j \in [L],$$

where $r_{i,j}$ is a long-term Diffie-Hellman shared key. Then, the clients send to the server the masked vector together with threshold encryptions of their pairwise mask.

$$\begin{aligned} \mathbf{y}_{i,t} &:= \mathbf{x}_{i,t} + \sum_{j \in [L]} \text{PRG}(h_{i,j,t}), \\ \{\text{ct}_{i,j}^{(0)}, \widehat{\text{ct}}_{i,j}^{(0)} &:= H(k_{i,j}^{(0)}) \oplus h_{i,j,t}\}_{j \in [L]} \end{aligned}$$

where $(k_{i,j}^{(0)}, \text{ct}_{i,j}^{(0)}) \leftarrow \text{TBIBKEM}.\text{Encap}(\text{pk}, t, j \| 0)$, for each $j \in [L]$. Moreover, we view the output of PRG as a bitstring and the sum operation is bit-wise.

– **Report-Decryptor**:

1. $\{\mathcal{S} \rightarrow \mathcal{D}_j\}_{j \in [L]}$: The server sends to the committee a vector with respect to the clients who contributed to aggregation at the current iteration, i.e.,

$$\mathbf{b} := (b_1, \dots, b_n) \text{ where } b_i = \begin{cases} 1, & \text{if } \mathcal{C}_i \text{ is online} \\ 0, & \text{if } \mathcal{C}_i \text{ is offline} \end{cases}$$

2. $\{\mathcal{D}_j \rightarrow \mathcal{S}\}_{j \in [L]}$: Each decryptor $\mathcal{D}_j$ locally derives the pairwise masks $h_{i,j,t}$ with all the clients according to the status vector $\mathbf{b}$ and also samples a fresh seed $m_{j,t} \leftarrow \$ \{0, 1\}^\lambda$. Then, it computes $\sum_{i \in S_t} \text{PRG}(h_{i,j,t})$ for all the online clients $\{\mathcal{C}_i\}_{i \in S_t}$ and masks its input vector as follows before sending it to the server,

$$\begin{aligned} \mathbf{y}_{j,t} &:= \mathbf{x}_{j,t} + \sum_{i \in S_t} \text{PRG}(h_{i,j,t}) + \text{PRG}(m_{j,t}) \\ \text{ct}_i^{(1)}, \widehat{\text{ct}}_j^{(1)} &:= H(k_i^{(1)}) \oplus m_{j,t}, \end{aligned}$$

where $(k_i^{(1)}, \text{ct}_i^{(1)}) \leftarrow \text{TBIBKEM}.\text{Encap}(\text{pk}, t, j \| 1)$.

– **Cross-check**:

1. $\{\mathcal{S} \rightarrow \mathcal{D}_j\}_{j \in [L]}$: The server computes an online-offline bit vector $\mathbf{b}' = (b'_1, \dots, b'_L)$ where $b'_j = 1$ if decryptor j responded in the report phase (i.e., is online) and $b'_j = 0$ otherwise (i.e., is offline). Then, it sends to the decryptors

$$\begin{aligned} \text{cm} &\leftarrow \text{TBIBKEM}.\text{Digest}(\text{crs}_{\text{TBIBKEM}}, \{(\text{id}_1 \| b'_1), \dots, (\text{id}_L \| b'_L)\}), \\ \pi &\leftarrow \text{SNARK}.\text{Prove}(\text{crs}_{\text{SNARK}}, (\text{cm}, (1 - \delta_D)L); (b'_1, \dots, b'_L)), \end{aligned}$$

where the SNARK proves that cm is a KZG vector commitment for the binary vector $\mathbf{b}'$ of length L and, further, at least $(1 - \delta_D)L$ decryptors are online (the relation is identical to $\mathcal{R}_{\text{GraphStatus}}$ given in Appendix C, except without the constraint from Equation (4)).

2. $\{\mathcal{D}_j \rightarrow \mathcal{S}\}_{j \in [L]}$: If $\text{SNARK.Vf}(\text{crSSNARK}, (\text{cm}, (1 - \delta_D)L)) = 0$, then abort. Else, send the partial decapsulations to the server:

$$\text{pd}_j \leftarrow \text{TBIBKEM.BatchPartDec}(\text{sk}_j, \text{cm}, t).$$

A threshold number of online committee members can collectively recover the pairwise masks of any missing committee member with all the (online) clients.

- **Reconstruction:** In a similar vein to the reconstruction step of the protocol in Section 4.2, the server reconstructs all the proper $k_i^{(1)}, k_{i,j}^{(0)}$ using the partial decryptions pd_j . Then, for any offline decryptor $\mathcal{D}_j$, the server retrieves pairwise masks $h_{i,j,t}$ with all the online clients and for any online decryptor $\mathcal{D}_j$, the server retrieves individual mask $m_{j,t}$. Then, it computes $\sum_{i \in S_t, j \in [L]} \text{PRG}(h_{i,j,t}) + \text{PRG}(m_{j,t})$ to derive the final aggregation $\sum \mathbf{x}_t$.

5.2 Security Analysis

We denote the full protocol by Φ_T , which consists of a sequential execution of Setup followed by T -round execution of Collection and Reconstruction, as described in Section 5.1. Next, we argue that Φ_T securely realizes $\mathcal{F}_{\text{Agg}}$ (Figure 2 with the blue lines).

Theorem 4. *Let $\delta, \delta_D, \eta, \eta_D$ be the threat model parameters. Let N be the total number of clients, n be the number of clients for summation in each round and L be the size of the decryptor committee. Assuming the existence of a PKI, a trusted source of initial randomness, a hash function H and a PRG (both modeled as a random oracle), a secure PRF, a succinct non-interactive argument of knowledge (SNARK) and a secure threshold batched identity-based key encapsulation mechanism (TB-IB-KEM), if $2\delta_D + \eta_D < 1/3$, then the protocol Φ_T securely realizes $\mathcal{F}_{\text{Agg}}$ in the random oracle model and in the presence of a malicious adversary with static (per round) corruptions controlling η fraction of N clients (and the server), except with probability at most $Tn \cdot 2^{-\kappa+1} + \text{negl}(\lambda)$.*

The proof of this theorem follows quite closely the proof of Theorem 3 given in Section 4.3, and therefore, we refrain from presenting the formal proof. Concretely, the simulation of the honest clients and decryptors proceeds as in the proof of Theorem 3, and the only difference is that during the report phase we need to additionally simulate the honest decryptors. However, since the honest decryptors in the report phase of the dynamic construction (partially) act as the clients in the static construction, these honest decryptors can be simulated analogously to the honest client in the proof of Theorem 3. Moreover, we want to point out that the ideal functionality $\mathcal{F}_{\text{Agg}}$ for the dynamic case (Figure 2 with the blue lines) also models static corruptions, as the newly corrupted parties are chosen at the beginning of each iteration.

Next, we discuss two important remarks regarding our dynamic construction.

Remark 1. *We emphasize that enforcing view consistency on the client status vector $\mathbf{b}$ remains unnecessary in our dynamic extension. Consider the scenario where the server misreports the set of active clients to the committee. If the server falsely labels an offline client i as online, the committee members will erroneously include the corresponding pairwise masks $\text{PRG}(h_{i,j})$ in their messages (i.e., partial decryption shares) to server. Since the client's payload $\mathbf{y}_i$ is effectively missing, these masks remain uncanceled in the final aggregate, rendering the result information-theoretically indistinguishable from random noise. This essentially satisfies the security with abort guarantee. Conversely, if the server incorrectly labels an online client i as offline (a form of censorship [BCGL⁺25]), the committee will exclude the necessary unmasking terms for that client.*

Consequently, the server is mathematically forced to discard i -th client’s input y_i entirely; attempting to include it without the committee’s matching shares would only corrupt the global sum with uncanceled noise. Thus, the server cannot selectively decrypt a target user’s input—it can only choose to include or exclude them from the aggregate entirely.

Remark 2. In the protocol presented in Section 5.1, committee members (i.e., decryptors) also contribute to the aggregation by masking their input vectors x_j using both pairwise masks (established with online clients) and an individual mask. The necessity for this double-masking mechanism is analogous to that of the clients in the static variant: in the event of a delayed report by a decryptor D_j , the server may label it as offline to legitimately retrieve the pairwise keys $\{h_{i,j}\}$ needed to recover the client aggregation. Without a secondary layer of protection, this key release would expose D_j ’s private input x_j to the server after it receives the masked input vector. The individual mask m_j thus serves as the requisite fallback protection. We note, however, that committee participation in the data aggregation is optional; decryptors may alternatively restrict their role to facilitating the protocol by sending only the partial decryption shares $\sum_{i \in S_t} \text{PRG}(h_{i,j})$ —excluding their own data vectors—thereby obviating the need for individual masks.

6 Performance Evaluation

6.1 Experiment Setup

We implement our protocol using Python and Rust. We follow the Flamingo prototype framework⁹ to build an FL training system with a secure aggregation layer. All experiments are conducted on a MacBook Pro with an Apple M4 Max CPU (16 cores) and 128 GB of unified memory. For the FL training components (e.g., gradient aggregation and vector computation), we follow Flamingo’s default settings to run all benchmarks. Following BBGLR and Flamingo, we adopt the ABIDES¹⁰ simulation to model real-world network connections. ABIDES incorporates a latency model defined by a base delay and jitter, which determines message arrival timing.

Following Flamingo, in our setting of benchmarks, the inputs of clients are 16K-dimensional vectors with 32-bit entries. The number of clients varies from 64 to 1024. The size of the committee (i.e., the number of decryptors) is fixed to be 60. The dropout rate is set as 10%. Note that we limit our evaluation to a direct and detailed comparison with Flamingo [MWA⁺23], as it shares our specific setting and threat model. Nevertheless, our protocol’s constant server-committee communication and computation complexity demonstrate a clear efficiency advantage over the state of the art, as shown in Table 1.

Implementation of threshold batched identity-based encryption scheme. We implement the variant of threshold batched encryption scheme introduced in [AFP25]. To integrate the Rust cryptographic library with our Python-based simulation framework, we use PyO3¹¹, a Rust binding framework that enables interoperability between Rust and Python. We also use maturin, a build tool for creating Python extensions from Rust codebases, for Rust and Python integration.

Specifically, the Rust implementation of batched threshold encryption provides core cryptographic functions, including key generation, encryption, partial decryption, and batch decryption operations. These operations are exposed to Python through PyO3’s `#[pyclass]` and `#[pyfunction]` macros, which automatically generate the necessary Python bindings. In the building process, we use maturin to compile the Rust code into a Python extension module (`batch-threshold`), which can be imported

⁹<https://github.com/eniac/flamingo>

¹⁰<https://github.com/abides-sim/abides>

¹¹<https://github.com/PyO3/pyo3>

directly in Python as a native module. We adopt this design to ensure that computationally intensive cryptographic operations (e.g., pairing computations, elliptic curve operations) are executed in Rust to optimize performance, while the high-level protocol logic remains in Python to maintain development flexibility.

Implementation of SNARKs. In our implementation, to compare with Flamingo, we consider two variants: one incorporating SNARKs and the other without SNARKs. The two variants are identical in all aspects except during the cross-check phase, where in the non-SNARK variant, the server explicitly transmits the status vectors of online and offline clients to the decryptors (i.e., following the Flamingo approach, see Appendix B). The SNARK-based variant follows the protocol described in Section 4.2.

We first use Groth16 as our SNARK system [Gro16]. Our implementation is based on the arkworks ecosystem¹², specifically using ark-groth16 with the BLS12-381 elliptic-curve pairing. We also implement a PLONK-based variant of our SNARK system [GWC19] for the cross-checking proof. Our PLONK implementation is based on the Jellyfish library¹³, which builds on the arkworks ecosystem using jf-plonk with the BLS12-381 elliptic-curve pairing. Similar to the batched threshold encryption integration, we use PyO3 and maturin to generate Python bindings for the Rust-based SNARK implementation. We use the method described in Appendix C to build the constraints for the cross-checking SNARK proof. Both Groth16 and PLONK variants leverage parallel computation for efficiency (e.g., parallel Multi-Scalar Multiplication (MSM) and Fast Fourier Transform (FFT) operations), enabled through the `parallel` feature flags in the arkworks and Jellyfish crates.

6.2 Communication Costs

We benchmark a single summation with 1,024 clients, where up to 10% of the clients may drop out at any given step (i.e., 10% dropout rate). Table 3 shows the benchmark results. The numbers demonstrate that our protocol significantly reduces communication overhead compared to Flamingo in a single summation over 1,024 clients. The overall communication is dominated by server-side overhead, where Flamingo sends 8,460.65 KB per aggregation due to expensive crosscheck and reconstruction phases. In contrast, our protocol reduces total server communication to 227.82 KB without SNARKs and 68.40 KB with SNARKs, a considerable $37\times$ and $124\times$ improvement respectively. Moreover, Flamingo requires each decryptor to send 44.24 KB during the crosscheck and reconstruction phases, whereas decryptors in our protocol send only 0.14 KB, a considerable reduction of $316\times$.

In addition, as shown in the Flamingo protocol (see Appendix B), the total communication cost during the reconstruction phase in Flamingo depends on the number of clients. Specifically, the reconstruction phase requires sending pairwise ciphertexts for each offline client and individual (masking) ciphertexts for online clients, which scales with the number of clients. In contrast, our solutions are independent of the number of clients, as the SNARK proof and verification key encode the online/offline status implicitly through the binary vector. Given that in circuit-specific SNARK systems like Groth16 the size of the verification key depends on the circuit size, server-to-committee communication may be slightly affected by the number of clients and their connected edges in the communication graph. We can also observe the same trend in Figure 5, which shows the communication costs between the server and committee members as the number of clients scales. In particular, the committee-to-server communication cost in our protocol is nearly constant.

¹²<https://github.com/arkworks-rs>

¹³<https://github.com/EspressoSystems/jellyfish>

Protocol	Server						
	Time (sec)			Comm. (KB)			
	Report	Crosscheck	Recon	Report	Crosscheck	Recon	Total Send
Flamingo	0.020	2.04	2.53	0.00	177.64	8283.01	8460.65
Lighthouse (without SNARKs)	0.006	2.43	6.03	0.00	177.82	50.00	227.82
Lighthouse (with Groth16)	0.005	3.00	6.07	0.00	18.40	50.00	68.40
Lighthouse (with PLONK)	0.005	5.50	6.21	0.00	113.85	50.00	163.85

Protocol	Client – Regular Clients						
	Time (sec)			Comm. (KB)			
	Report	Crosscheck	Recon	Report	Crosscheck	Recon	Total Send
Flamingo	0.080	—	—	75.41	—	0.00	75.41
Lighthouse (without SNARKs)	0.129	—	—	70.76	—	—	70.76
Lighthouse (with Groth16)	0.128	—	—	70.76	—	—	70.76
Lighthouse (with PLONK)	0.131	—	—	70.76	—	—	70.76

Protocol	Client – Decryptors						
	Time (sec)			Comm. (KB)			
	Report	Crosscheck	Recon	Report	Crosscheck	Recon	Total Send
Flamingo	—	1.83	0.026	—	0.27	43.97	44.24
Lighthouse (without SNARKs)	—	1.85	—	—	0.14	—	0.14
Lighthouse (with Groth16)	—	0.003	—	—	0.14	—	0.14
Lighthouse (with PLONK)	—	0.004	—	—	0.14	—	0.14

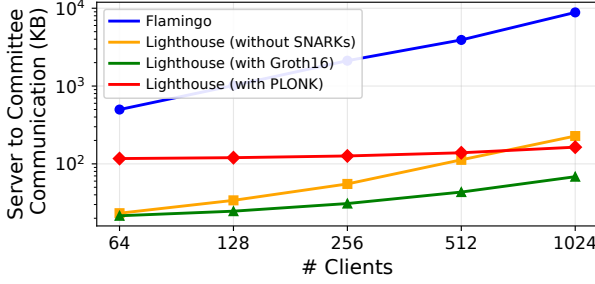
Table 3: Microbenchmarks averaged over multiple iterations for a single aggregation over 1024 clients. Times are mean seconds per iteration excluding setup. The Communication costs report send volume only. Total Send is the sum of send costs across report, crosscheck, and reconstruction. Server communication is reported as total bytes per iteration. Client communication is reported per regular client or per decryptor, respectively.

6.3 Computation Costs

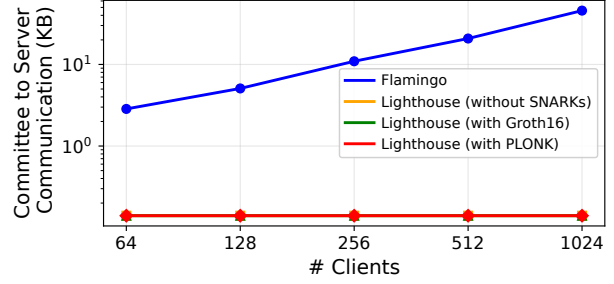
Table 3 also shows the comparison of computation costs. We observe that decryptors in our protocol with SNARKs spend only 0.004 s in the crosscheck phase, which is much faster than in Flamingo. This is because the decryptors only need to verify the SNARK proof rather than generate the graph G_t to verify the cross-check conditions (see Appendix B). Specifically, SNARK verification in Groth16 requires only $O(1)$ constant-time pairing operations, independent of the number of clients, whereas Flamingo’s approach requires constructing and verifying the graph connectivity.

On the server side, the expensive components of our approach are generating SNARK proofs and reconstruction. SNARK proof generation depends on the structure of the graph G_t , as the circuit must verify constraints for each client and edge. However, we argue that this overhead is acceptable in practice and can be further optimized through parallel computation (MSM and FFT operations), as FL servers are typically provisioned with powerful hardware. The reconstruction phase involves batch decryption operations, which scale with the number of online clients and offline client pairs, but this is comparable to Flamingo’s reconstruction time.

Also, Figure 6 shows that the total computation time of a decryptor remains constant, as besides the SNARKs verification, decryptors only need to perform `TBIBKEM.BatchPartDec` on their private key sk_k , the commitment cm , and the iteration number t , all of which are independent of the number of clients. This is a significant advantage over Flamingo, where decryptors’ computation time increases with the number of clients.

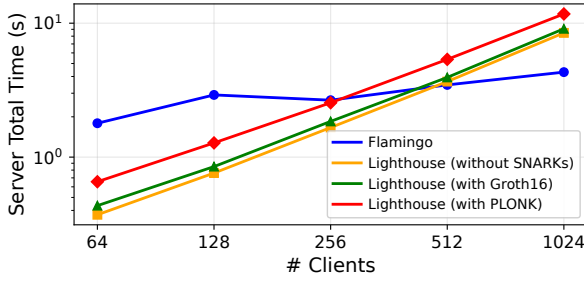


(a) Server to committee communication costs

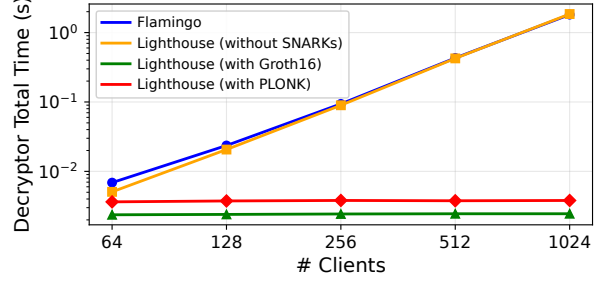


(b) Committee to server communication costs

Figure 5: The communication costs between the server and committee members (i.e., decryptors).



(a) Server total computation time



(b) Total computation time per-decryptor

Figure 6: The total computation time of the server and committee members (i.e., decryptors).

7 Related Work

Single-server setting. Majority of the secure aggregation protocols fall into this category where a single (untrusted) server interacts with a large number of clients to compute the aggregation, typically over a number of aggregation iterations. Starting from the early work by Bonawitz et al. [BIK⁺17], substantial advancements have been made to reduce the overheads at the clients and server [BBG⁺20, LLPT23, MWA⁺23, MLLP25], efficiently tolerating client’s drop-out [MWA⁺23, vKLMS25] and support for dynamic participation [BCGL⁺25, KP25]. Existing single-server secure aggregation protocols typically follow two blueprints of private pairwise-masking [BIK⁺17, BBG⁺20, MWA⁺23, vKLMS25] and additively homomorphic encryption [LLPT23, BCGL⁺25, KP25]. In the former, each client should properly mask their input vector such that the masks collectively cancel out and server obtains the final aggregation result. Using such one-time pad masking is the most efficient approach for encrypting large-input vectors (as typically needed in FL settings). In this model, however, a crucial challenge is to provide resilience against client’s drop out that may render the final sum fail to obtain. Seminal works by Bonawitz et al. [BIK⁺17] and Bell et al. [BBG⁺20] proposed double-masking techniques to protect users against both the central server and collusion among other clients. They also addressed the dropout issue using the idea of sharing the mask seed with peers for collective recovery, with a quadratic communication cost for the fully connected graph setting of [BIK⁺17] and a linearithmic cost for the the log n -sparse graph setting of [BBG⁺20], where each client connects to only $\log n$ neighbors. Flamingo [MWA⁺23] improved upon these efforts to further deploy a committee of clients (i.e., decryptors) to handle decryptions/recovery of the masks. They also introduced the concept of reusable secrets with a one-time setup phase, allowing to derive fresh seeds (via a PRF) for use in subsequent rounds without the need for repeated key generations. This “setup-once, run-

many” approach dramatically amortizes the communication cost over the duration of the training session. The use of such small committee helping with the deriving the result since then has become a vital design choice also in constructions based on homomorphic aggregation [BCGL⁺25, KP25]. Intuitively, these protocols work by having the clients encrypt their input vector under the public key of the committee and then let the server (or committee) compute the aggregated ciphertext to be collectively decrypted. The main bottleneck here is incurring high computational overhead, particularly for large-input vectors in practice where we consider in this work. LERNA [LLPT23] and OPA [KP25] require each client to secret-share their keys with the committee for their LWE-based key homomorphic masking schemes. This property allows the aggregation of masks to be performed in the key space rather than the mask space. If the committee can securely aggregate the secret keys of the participating users ($K = \sum k_i$), they can generate a single aggregate mask $F_K(x)$ that unmask the sum of the user inputs. This decouples the generation of the mask from the participation and thus input vector of the users. That is, the user contributes their key share to the committee and their masked input to the server. Despite their elegant solutions, OPA [KP25] does not consider the problem of maintaining aggregation integrity when the server is malicious and may incorporate a client’s contribution multiple times to bias the result. In contrast, Willow [BCGL⁺25] seeks to prevent a malicious server from incorporating a client’s contribution multiple times. To this end, Willow introduces an additional entity, referred to as an auxiliary verifier committee, whose role is to certify that each local input is added to the aggregation buffer by the server at most once, causing a linear $O(n)$ overhead.

Crucially, while OPA and Willow achieve low asymptotic decryptor communication, neither reaches the optimal $O(1)$ efficiency of our protocol. Furthermore, both protocols rely heavily on lattice-based cryptography, which suffers significant concrete performance penalties when handling the high-dimensional vectors typical in FL workloads (e.g., 16K parameters of 32-bit elements). For instance, Willow requires 227 seconds of server computation for 1,000 clients—roughly an order of magnitude slower than our protocol’s server runtime for the same cohort size (albeit measured on different hardware architectures). Moreover, we acknowledge the recent concurrent work WillowFold [HAG⁺26], which optimizes Willow by employing a recursive folding tree. This elegant approach introduces a novel “streaming aggregation” property and reduces server-to-committee communication to a logarithmic $O(\log n)$ overhead. While it reduces the decryptor cost compared to the baseline Willow protocol, it still falls short of the strict, optimal $O(1)$ efficiency achieved by our laconic (SNARK-based) construction. Our work Lighthouse builds on the single-server paradigm but targets the specific problem of making server-committee communication succinct, avoiding the inefficiencies of high-dimensional lattice constructions, and supporting dynamic client participation, where new clients can join at any time, which is not supported by most previous works [MWA⁺23, MLLP25, LLPT23].

Multi-server setting. Trust can be distributed across multiple aggregation servers to achieve lightweight efficiency. Prio [CGB17] and Prio+ [AGJ⁺22] use secret sharing to split client data among non-colluding servers, enabling high throughput. Heli [LCGDW26] builds upon the two-server architecture of Prio, introducing an asymmetric design where the “Heavy” server performs work linear in the number of clients ($O(n)$), while the “Light” server’s work is sublinear, scaling primarily with the number of dropouts. In particular, input values are restricted to small integers since decryption necessitates solving a discrete logarithm instance. ELSA [RSWP23] improves upon this by allowing clients to act as “untrusted dealers” of cryptographic correlations, significantly reducing the interactive overhead between servers. However, these systems traditionally rely on a strong semi-honest or any-trust assumption. Addressing this, Mario [NLT25] introduces a robust multi-aggregator framework that remains secure even when a subset of servers is malicious, albeit at increased computational cost. While multi-server systems aim at offering better performance compared to the ones with single-server, they inherently depend on the existence of

distinct, non-colluding administrative domains—an assumption often untenable in cross-silo deployments.

Robust protocols. A parallel line of research attempt at providing robustness that is two-fold in secure aggregation: resilience to client dropouts (liveness) and defense against active adversaries (safety). To do that latter, they use various input validation mechanisms, where each client has to prove that they follow the protocol and submit a well-formed input vector. To prevent model poisoning by malicious clients, ACORN [BGL⁺23] enables the server to verify that aggregated inputs satisfy specific norms (L_2, L_∞) using Zero-Knowledge Proofs (ZKPs). Armadillo [MGKP25] extends this line of work by combining input validation with robust aggregation mechanics to ensure valid models can be computed even under adversarial conditions. Particularly, they introduced techniques to provide *disruption resistance*, where any coalition of malicious clients can affect the aggregation result only by misreporting their private inputs in a pre-defined legitimate range. Particular noteworthy is that Armadillo achieves robustness in 3 rounds, matching the round complexity of non-robust protocols like Flamingo [MWA⁺23], whereas previous robust protocols (like ACORN) required significantly more interaction. We leave enhancing our protocol with robustness techniques as an interesting future work.

Acknowledgments

We would like to thank Yiping Ma and Varun Madathil for insightful discussions on Flamingo [MWA⁺23] and TACITA [MLLP25]. The work of the first, third and fourth authors are supported in part by the AFOSR Award FA9550-24-1-0156, and research grants from the Bakar Fund, Peder Sather Funds, the Stellar Development Foundation, Supra Inc., Category Labs, Input Output Global, and Byzantine Research Inc. The work of the fourth author is additionally supported by the Austrian Science Fund (FWF) Project J4879-N. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of these institutes.

References

- [ABD⁺25] Amit Agarwal, Kushal Babel, Sourav Das, Babak Poorebrahim Gilkalaye, Arup Mondal, Benny Pinkas, Peter Rindal, and Aayush Yadav. Weighted batched threshold encryption with applications to mempool privacy. *Cryptology ePrint Archive*, 2025. 12
- [ABIW22] Sebastian Angel, Andrew J Blumberg, Eleftherios Ioannidis, and Jess Woods. Efficient representation of numerical optimization problems for {SNARKs}. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 4273–4290, 2022. 15
- [AFP25] Amit Agarwal, Rex Fernando, and Benny Pinkas. Efficiently-thresholdizable batched identity based encryption, with applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 69–100. Springer, Cham, August 2025. 4, 8, 12, 17, 18, 32
- [AGJ⁺22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *International Conference on Security and Cryptography for Networks*, pages 516–539. Springer, 2022. 36
- [AP19] Diego F. Aranha and Elena Pagnin. The simplest multi-key linearly homomorphic signature scheme. In Peter Schwabe and Nicolas Thériault, editors, *LATINCRYPT 2019*, volume 11774 of *LNCS*, pages 280–300. Springer, Cham, October 2019. 5

- [BBG⁺20] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrede Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1253–1269. ACM Press, November 2020. [3](#), [4](#), [28](#), [35](#), [43](#), [44](#)
- [BCF⁺25] Jan Bormet, Arka Rai Choudhuri, Sebastian Faust, Sanjam Garg, Hussien Othman, Guru-Vamsi Policharla, Ziyang Qu, and Mingyuan Wang. BEAST-MEV: Batched threshold encryption with silent setup for MEV prevention. *Cryptology ePrint Archive*, Report 2025/1419, 2025. [12](#)
- [BCGL⁺25] James Bell-Clark, Adrià Gascón, Baiyu Li, Mariana Raykova, and Philipp Schoppmann. Willow: Secure aggregation with one-shot clients. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VIII*, volume 16007 of *LNCS*, pages 285–318. Springer, Cham, August 2025. [3](#), [5](#), [26](#), [27](#), [28](#), [31](#), [35](#), [36](#)
- [BFOQ25] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyang Qu. BEAT-MEV: Epochless approach to batched threshold encryption for MEV prevention. In Lujo Bauer and Giancarlo Pellegrino, editors, *USENIX Security 2025*, pages 3457–3476. USENIX Association, August 2025. [8](#), [12](#)
- [BGL⁺23] James Bell, Adrià Gascón, Tancrede 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, 2023. [15](#), [37](#)
- [BIK⁺17] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahon, 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, *ACM CCS 2017*, pages 1175–1191. ACM Press, October / November 2017. [3](#), [6](#), [22](#), [26](#), [35](#)
- [BLT25] Dan Boneh, Evan Laufer, and Ertem Nusret Tas. Batch decryption without epochs and its application to encrypted mempools. *Cryptology ePrint Archive*, Report 2025/1254, 2025. [12](#)
- [CDG⁺17] Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Cham, August 2017. [8](#)
- [CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 55–72. Springer, Berlin, Heidelberg, February / March 2013. [11](#)
- [CGB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*, pages 259–282, 2017. [36](#)
- [CGPP24] Arka Rai Choudhuri, Sanjam Garg, Julien Piet, and Guru-Vamsi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024. [4](#), [8](#), [11](#)
- [CGPW25] Arka Rai Choudhuri, Sanjam Garg, Guru Vamsi Policharla, and Mingyuan Wang. Practical mempool privacy via one-time setup batched threshold encryption. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3477–3495, 2025. [4](#)

- [DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, New York, August 1990. [6](#), [11](#), [44](#)
- [FK23] Dankrad Feist and Dmitry Khovratovich. Fast amortized KZG proofs. *Cryptology ePrint Archive*, Report 2023/033, 2023. [11](#)
- [GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013. [7](#)
- [GHK⁺25] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and applications. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 504–539. Springer, Cham, August 2025. [4](#), [8](#)
- [GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Berlin, Heidelberg, May 1999. [5](#), [15](#), [44](#)
- [GJM⁺24] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hinTS: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy*, pages 3034–3052. IEEE Computer Society Press, May 2024. [5](#)
- [GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 352–386. Springer, Cham, August 2024. [4](#), [5](#), [8](#)
- [Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 305–326. Springer, 2016. [33](#)
- [GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, Report 2019/953, 2019. [33](#)
- [GWWW25] Junqing Gong, Brent Waters, Hoeteck Wee, and David J Wu. Threshold batched identity-based encryption from pairings in the plain model. *Cryptology ePrint Archive*, 2025. [6](#), [8](#), [27](#)
- [HAG⁺26] Hossein Hafezi, Kasra Abbaszadeh, Adrià Gascón, Phillipp Schoppmann, Mariana Raykova, and Benedikt Bünz. Willowfold: Secure aggregation with a lightweight committee. *Cryptology ePrint Archive*, 2026. [5](#), [36](#)
- [HJKS15] Felix Heuer, Tibor Jager, Eike Kiltz, and Sven Schäge. On the selective opening security of practical public-key encryption schemes. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 27–51. Springer, Berlin, Heidelberg, March / April 2015. [9](#)
- [HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *Annual Cryptology Conference*, pages 132–150. Springer, 2011. [3](#)

- [HRW16] Dennis Hofheinz, Vanishree Rao, and Daniel Wichs. Standard security does not imply indistinguishability under selective opening. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 121–145. Springer, Berlin, Heidelberg, October / November 2016. [42](#)
- [HSBB25] Hossein Hafezi, Alireza Shirzad, Benedikt Bünz, and Joseph Bonneau. IronDict: Transparent dictionaries from polynomial commitments. *Cryptology ePrint Archive*, Report 2025/1580, 2025. [15](#)
- [HW24] Justin Holmgren and Brent Waters. Separating selective opening security from standard security, assuming IO. *Cryptology ePrint Archive*, Report 2024/1080, 2024. [42](#)
- [KL07] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, 2007. [18](#)
- [KMA⁺21] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021. [3](#)
- [KP25] Harish Karthikeyan and Antigoni Polychroniadou. OPA: One-shot private aggregation with single client interaction and its applications to federated learning. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part VIII*, volume 16007 of *LNCS*, pages 319–353. Springer, Cham, August 2025. [3](#), [4](#), [5](#), [15](#), [35](#), [36](#)
- [KWJ24] Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. Sok: public randomness. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*, pages 216–234. IEEE, 2024. [14](#), [44](#)
- [KYR⁺24] Nikolas Koutsoubis, Yasin Yilmaz, Ravi P Ramachandran, Matthew Schabath, and Ghulam Rasool. Privacy preserving federated learning in medical imaging with uncertainty estimation. *arXiv preprint arXiv:2406.12815*, 2024. [5](#)
- [KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 177–194. Springer, Berlin, Heidelberg, December 2010. [8](#), [10](#)
- [LCG⁺24] Julia Len, Melissa Chase, Esha Ghosh, Kim Laine, and Radames Cruz Moreno. OPTIKS: An optimized key transparency system. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024. [15](#)
- [LCGDW26] Ryan Lehmkuhl, Henry Corrigan-Gibbs, Emma Dauterman, and David J Wu. Heli: Heavy-light private aggregation. *Cryptology ePrint Archive*, 2026. [36](#)
- [LLPT23] 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, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 302–334. Springer, Singapore, December 2023. [5](#), [35](#), [36](#)
- [LPS25] Helger Lipmaa, Roberto Parisella, and Janno Siim. On knowledge-soundness of plonk in rom from falsifiable assumptions. In *Annual International Cryptology Conference*, pages 362–395, 2025. [27](#)

- [MBB⁺15] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. {CONIKS}: Bringing key transparency to end users. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 383–398, 2015. [15](#)
- [MGKP25] Yiping Ma, Yue Guo, Harish Karthikeyan, and Antigoni Polychroniadou. Armadillo: Robust single-server secure aggregation for federated learning with input validation. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 2219–2233, 2025. [14](#), [15](#), [37](#)
- [MLLP25] Varun Madathil, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. Tacita: Threshold aggregation without client interaction. *Cryptology ePrint Archive*, 2025. [4](#), [5](#), [9](#), [15](#), [27](#), [35](#), [36](#), [37](#), [42](#), [43](#)
- [MMR⁺17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017. [3](#)
- [MSDS19] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy*, pages 691–706. IEEE Computer Society Press, May 2019. [3](#)
- [MWA⁺23] 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. [3](#), [4](#), [5](#), [6](#), [7](#), [9](#), [13](#), [14](#), [15](#), [19](#), [21](#), [22](#), [26](#), [27](#), [28](#), [32](#), [35](#), [36](#), [37](#), [42](#), [43](#)
- [NLT25] Truong Son Nguyen, Tancrede Lepoint, and Ni Trieu. Mario: multi-round multiple-aggregator secure aggregation with robustness against malicious actors. In *2025 IEEE 10th European Symposium on Security and Privacy (EuroS&P)*, pages 1140–1164. IEEE, 2025. [36](#)
- [RSWP23] 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. IEEE, 2023. [36](#)
- [TAB⁺20] Alin Tomescu, Ittai Abraham, Vitalik Buterin, Justin Drake, Dankrad Feist, and Dmitry Khovratovich. Aggregatable subvector commitments for stateless cryptocurrencies. In Clemente Galdi and Vladimir Kolesnikov, editors, *SCN 20*, volume 12238 of *LNCS*, pages 45–64. Springer, Cham, September 2020. [8](#), [11](#)
- [TBP⁺19] Alin Tomescu, Vivek Bhupatiraju, Dimitrios Papadopoulos, Charalampos Papamanthou, Nikos Triandopoulos, and Srinivas Devadas. Transparency logs via append-only authenticated dictionaries. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 1299–1316. ACM Press, November 2019. [15](#)
- [TKPS22] Ioanna Tzialla, Abhiram Kothapalli, Bryan Parno, and Srinath T. V. Setty. Transparency dictionaries with succinct proofs of correct operation. In *NDSS 2022*. The Internet Society, April 2022. [15](#)
- [vKLMS25] Elina van Kempen, Qifei Li, Giorgia Azzurra Marson, and Claudio Soriente. A lightweight secure aggregation protocol for federated learning applications. In *2025 IEEE 45th International Conference on Distributed Computing Systems (ICDCS)*, pages 23–33. IEEE, 2025. [3](#), [5](#), [35](#)

- [ZLWL25] Handi Zhang, Langchen Liu, Kangyu Weng, and Lu Lu. Federated scientific machine learning for approximating functions and solving differential equations with data heterogeneity. *IEEE Transactions on Neural Networks and Learning Systems*, 2025. 5
- [ZOZ25] Tianyu Zhang, Yupeng Ouyang, and Yupeng Zhang. Dynark: Making Groth16 dynamic. Cryptology ePrint Archive, Report 2025/1897, 2025. 11

A Oversight in Prior Constructions

We describe here mistakes that appear in the security proofs of prior works, namely Flamingo [MWA⁺23] and TACITA [MLLP25]. We have brought this to the attention of the authors of these works and they have acknowledged it. We remark that these are security proof issues that invalidate their theorem statements rather than concrete attacks against their protocols. Thus, following our proposed approach should potentially allow for a way to fix.

The core issue in both works stems from the following observation. A common theme in committee-based secure aggregation protocols is to have clients first mask their inputs and encrypt these masks. Later, the server, with the help of a decryption committee, obtains the necessary masks to recover the aggregated result. In the security proof, we must construct a simulator that simulates the behavior of the honest clients and decryptors to the malicious server in an indistinguishable way. Since a majority of the decryptors are honest, the simulator must provide correct partial decryptions to the server for whichever ciphertexts (i.e., encrypted masks) the server chooses to decrypt. While an honest server would only request decryptions of the masks needed to recover the correct aggregation, a malicious server can request decryption of *any* subset of the masks from the committee—even subsets that do not allow it to recover the correct aggregation result. Hence, the simulator must be able to answer decryption queries for any subset of the ciphertexts. At the same time, for security we need to argue that a malicious server learns *only* the masks it explicitly requested, and nothing about the remaining masks. To establish this during the security proof, we must replace the ciphertexts encrypting the masks *outside* the adversary’s decryption request with encryptions of dummy values (e.g., encryptions of 0). Ideally, this replacement should follow from the security of the underlying encryption scheme, and the replaced ciphertexts should serve as challenge ciphertexts, since they are never decrypted.

However, as noted above, the server can request decryption of any subset of the ciphertexts, and moreover, it can do so *after* seeing all the honest clients’ ciphertexts. This creates a situation where the set of challenge ciphertexts — those that will *not* be decrypted — is not known a priori, yet the simulator must still be able to decrypt the subset chosen by the malicious server. Consequently, conventional security notions such as IND-CPA or IND-CCA are insufficient: in these notions, the challenge ciphertexts are fixed in advance and the adversary is explicitly forbidden from decrypting them.

To prove such protocols secure, we therefore need a stronger¹⁴ security notion for the underlying encryption scheme, namely *indistinguishability under selective opening attacks* (IND-SOA). In the IND-SOA security game, the adversary sees a batch of ciphertexts and is then allowed to “open” (i.e., request decryption of) an adaptively chosen subset. Security requires that the unopened ciphertexts do not leak any information to the adversary. This is precisely the notion needed to resolve the dilemma described above.

Returning to Flamingo [MWA⁺23] and TACITA [MLLP25], the mistakes in their security proofs are directly related to this issue. Specifically, the proof of [MWA⁺23, Theorem 3] assumes that the underlying threshold encryption scheme is only IND-CPA secure. As a result, the simulator in Hybrids 3 and 4 cannot consistently simulate the honest committee members, since it is unable to compute the necessary partial

¹⁴We remark that the IND-SOA security notion is strictly stronger than IND-CCA, as shown in [HRW16, HW24]. Moreover, we note that the approach we have used in Section 4.2 provides even stronger simulation-based selective opening (SIM-SO) security.

decryptions during the reconstruction step. Similarly, in [MLLP25, Theorem 2], the authors assume that the underlying silent threshold encryption scheme is (extended) IND-CPA secure, and therefore the simulation in Hybrid 6 fails for the same reason: the simulator cannot provide the required partial decryptions when simulating the honest decryptors. We note that upgrading to an IND-CCA secure encryption scheme would not resolve these issues either, since neither IND-CPA nor IND-CCA security allows decryption of *any* challenge ciphertext—unlike IND-SOA security, which grants the adversary selective decryption capability over the subset of challenge ciphertexts.

B The Flamingo Protocol

Flamingo [MWA⁺23] is a single-server secure aggregation protocol that allows computing the sum of a large set of clients' inputs sampled from a population of size N across multiple iterations while maintaining the privacy of the individual ones. It follows the masking-based blueprint of [BBG⁺20], where clients derive pairwise random masks to blind their input vectors such that when all the masked input vectors across all clients are added, they cancel out each other. Unlike the previous work that runs a fresh single-iteration protocol consecutively [BBG⁺20], Flamingo eliminates the need for per-iteration setup phase that makes it particularly suitable for the federated learning setting. Crucially, given that in federated learning settings clients are prone to drop-out due to various issues (e.g., network conditions, power loss and etc), Flamingo puts forth the idea of deploying a committee of clients (called decryptors) to help the server to remove the masks and derive the result. This design improves the run time over [BBG⁺20] by eliminating the need to involve all the clients for mask cancellation and instead shift the workload to a small set of decryptor. Finally, they introduced a technique to let each client generate a per-iteration graph which establishes a set of random neighbors per client to derive their pairwise masks. We discuss these ideas in more details as follows:

Multi-iteration aggregation without re-running setup. Flamingo separates long-lived pairwise secrets from per-iteration masking material. Each client i holds a long-term secret $r_{i,j} = g^{a_i a_j}$ with each neighbor j (which are Diffie-Hellman key exchanges between the client and its neighbors.). In iteration t , client i derives pairwise seeds by

$$h_{i,j,t} \leftarrow \text{PRF}(r_{i,j}, t) \quad \text{for all } j \in \mathcal{N}_t(i),$$

and also samples a fresh individual-mask seed $m_{i,t}$. Here, $\mathcal{N}_t(i)$ is the set of neighbors of i in the graph at the t -th iteration. These seeds are used to protect client' input privacy in the event of a delay in arriving the masked input to the server which leads to recovering the pairwise masks (see [MWA⁺23]). Masks are expanded as $\text{PRG}(h_{i,j,t})$ and $\text{PRG}(m_{i,t})$, and the submitted vector is

$$\begin{aligned} \mathbf{y}_{i,t} = & \mathbf{x}_{i,t} + \sum_{\substack{j \in \mathcal{N}_t(i) \\ j > i}} \text{PRG}(h_{i,j,t}) \\ & - \sum_{\substack{j \in \mathcal{N}_t(i) \\ j < i}} \text{PRG}(h_{i,j,t}) + \text{PRG}(m_{i,t}). \end{aligned}$$

with the $\pm$ signs chosen so pairwise masks cancel out in the sum. For convenience, we will henceforth use Flamingo's concise notation

$$\mathbf{y}_{i,t} := \mathbf{x}_{i,t} + \sum_{j \in \mathcal{N}_t(i)} \pm \text{PRG}(h_{i,j,t}) + \text{PRG}(m_{i,t}).$$

Crucially, only the per-iteration seeds $\{h_{i,j,t}\}_j$ and $m_{i,t}$ are encrypted and sent (under a threshold public key); the long-term $r_{i,j}$ are never revealed. This ensures iteration freshness and forward isolation: Even if a seed is recovered in iteration t , it does not help in any other iteration, and no party ever learns $r_{i,j}$ from which future seeds could be derived. Attempting to re-use setup as in [BBG⁺20] would leak masks across iterations: once the server reconstructs a pairwise secret for a dropout in one iteration, it can reuse it in later iterations. The upshot is a one-time amortized setup (plus a PKI and a randomness beacon) that supports changing cohorts S_t (the set of clients involved in iteration t) without re-keying each iteration.

Lightweight dropout-resilience. Flamingo handles client dropouts by attaching ciphertexts of the seeds to each masked type (pairwise or individual) and using threshold decryption to recover only what is needed—either the individual or the pairwise for online and offline clients respectively. Flamingo instantiates it with threshold ElGamal [DF90], where a seed s is encrypted as $(g^r, s \cdot y^r)$ with $y = g^{sk}$. Protocol logic ensures that for each client the server can recover *only* one of the two seed types (pairwise bundle versus individual seed), preserving privacy while still enabling unmasking when a client drops out. In particular, the server should learn the individual masks for online clients and the pairwise masks for the offline ones. To ensure secure and correct mask cancellation during reconstruction, the server and the committee must agree on the set of online/offline clients. This synchronization necessitates a dedicated communication round to establish a consistent view of the client status vector. Furthermore, to facilitate reconstruction the committee members must receive the full set of client ciphertexts from the server, causing a bottleneck for the committee to handle bandwidth proportional to the total number of clients. This thus yields a two-step, server-centric recovery path where clients submit in one-shot and the server later collects $2f + 1$ partial decryptions for their corresponding masks (depending on being offline or online). A key subtlety is generating the threshold key in a server-mediated star topology: Flamingo adapts a variant of popular Gennaro et al. Distributed Key Generation (DKG) [GJKR99] to accommodate the event of message dropout by the server.

Per-iteration graph establishment. In the federated learning setting, the specific subset of clients selected to participate varies across iterations. To ensure the masking graph remains connected—and thus secure—regardless of which subset is sampled, Flamingo avoids committing to a single static topology. Instead, it employs a mechanism where the connectivity graph G_t is logically regenerated for each aggregation iteration t based on the current cohort. Specifically, the topology is derived deterministically using a public randomness beacon [KWJ24] and the iteration identifier. This allows the active clients—who must already be registered in the system—to locally compute a fresh, sparse list of neighbours that guarantees connectivity for that specific round. By tuning a single density parameter γ , Flamingo ensures the per-iteration graph remains sufficiently connected for security with high probability, balancing robust masking against the need for low communication and computation overhead. For completeness, we next present the detailed Flamingo protocol steps.

Setup: It happens once.¹⁵

1. Distribute a random seed ν .
2. $\{\mathcal{S} \rightarrow \mathcal{D}_k\}_{k \in [L]}$: The server selects committee members $\mathcal{D}_1, \dots, \mathcal{D}_L$ at random.
3. $\{\mathcal{D}_k\}_{k \in [L]} \xrightarrow{\mathcal{S}} \{\mathcal{D}_k\}_{k \in [L]}$: A threshold ElGamal key-pair is generated pk and $(sk_1, \dots, sk_L)$ for each committee member i respectively.

¹⁵Except every R iterations the committee is updated and a secret key transfer protocol happens that involves the old and new committee members.

Collection: It proceeds in T iterations. In each iteration $t = 1, \dots, T$ we have:

- Report:

1. $\{\mathcal{C}_i \rightarrow \mathcal{S}\}_{i \in [n]}$: Each client i derives pairwise seeds by $h_{i,j,t} \leftarrow \text{PRF}(r_{i,j}, t)$ for all $j \in \mathcal{N}_t(i)$, where $r_{i,j}$ is a long-term Diffie-Hellman shared key. The clients send to the server the masked vector together with threshold encryptions of their masks.

$$\mathbf{y}_{i,t} = \mathbf{x}_{i,t} + \sum_{j \in \mathcal{A}_t(i)} \pm \text{PRG}(h_{i,j,t}) + \text{PRG}(m_{i,t})$$

$$\phi_i = \text{TEnc}(\text{pk}, m_{i,t})$$

$$\{\psi_{i,j} = \text{TEnc}(\text{pk}, h_{i,j,t})\}_{j \in \mathcal{N}_t(i)}$$

- Cross-check:

1. $\{\mathcal{S} \rightarrow \mathcal{D}_k\}_{k \in [L]}$: The server sends to the committee an 'online-offline' vector wrt the clients.

$$\mathbf{b} = (b_1, \dots, b_n) \text{ where } a_i = \begin{cases} 1, & \text{if } \mathcal{C}_i \text{ online} \\ 0, & \text{if } \mathcal{C}_i \text{ offline} \end{cases}$$

2. $\{\mathcal{D}_k \rightarrow \mathcal{S}\}_{k \in [L]}$: Each decryptor signs the 'online-offline' vector.

$$\sigma_k = \text{Sign}(\cdot, \mathbf{b})$$

3. $\{\mathcal{S} \rightarrow \mathcal{D}_k\}_{k \in [L]}$: The server sends back to each decryptor the other decryptors' signatures.

$$\sigma_1, \dots, \sigma_L$$

Each decryptor locally checks that: (I) at least $2L/3$ of the signatures are valid (wrt $\mathbf{b}$), (II) at least $(1 - \delta)n$ clients are online, (III) all online clients are connected and (IV) each online client has at least χ online client neighbors such that $\eta^\chi < 2^\kappa$. If not the decryptor aborts.

- Reconstruct:

1. $\{\mathcal{S} \rightarrow \mathcal{D}_k\}_{k \in [L]}$: The server sends to the committee the proper threshold ciphertexts, according to the 'offline-online' vector.

$$\mathbf{ct} = (\text{ct}_1, \dots, \text{ct}_n) \text{ where}$$

$$\text{ct}_i = \begin{cases} \phi_i, & \text{if } \mathcal{C}_i \text{ online} \\ \{\psi_{j,i}\}_{j \in \mathcal{N}_t(i)}, & \text{if } \mathcal{C}_i \text{ offline} \end{cases}$$

2. $\{\mathcal{D}_k \rightarrow \mathcal{S}\}_{k \in [L]}$: Each decryptor respond with their corresponding partial decryptions.

$$\mathbf{d} = (d_1, \dots, d_n) \text{ where}$$

$$d_i = \begin{cases} \text{PartDec}(\text{sk}_k, \phi_i), & \text{if } \mathcal{C}_i \text{ online} \\ \{\text{PartDec}(\text{sk}_k, \psi_{j,i})\}_{j \in \mathcal{N}_t(i)}, & \text{if } \mathcal{C}_i \text{ offline} \end{cases}$$

Reconstruction: The server reconstructs all the proper $m_{i,t}$, $h_{i,j,t}$ and adds accordingly with the $\mathbf{y}_{i,t}$'s to retrieve $\sum \mathbf{x}_{i,t}$.

Flamingo Asymptotic cost. The communication between the server and the committee of threshold decryptors in Flamingo is $O(n + L)$. Concretely each decryptor:

- Receives: $n \text{ bits} + (L - 1)|\sigma| + (1 - \delta)n|\text{ct}| + \delta\gamma n|\text{ct}|$.
- Sends: $|\sigma| + (1 - \delta)n|\text{pd}| + \delta\gamma n|\text{pd}|$.

where $|\text{ct}|$ is the size of a threshold encryption ciphertext, $|\text{pd}|$ is the size of the partial decryption, and γ is the upper bound on the number of neighbors of a client.

C Succinct Cross-checking Proof

Let $\mathbb{F}$ be a finite field with $\text{char}(\mathbb{F}) > n$. Let $G = (V, E)$ be a public, deterministically generated undirected graph with vertex set $V = \{1, \dots, n\}$. For each $i \in V$, let $\mathcal{N}(i) := \{j \in V \mid (i, j) \in E\}$ denote the public neighborhood of client i . Let $\alpha, \beta \in \mathbb{N}$ be public parameters denoting the minimum required number of online clients in V and the minimum required online neighbors for each active client, respectively.

NP Relation. We define the relation $\mathcal{R}_{\text{GraphStatus}}$ for the instance $\mathbb{x} = (\text{cm}, G, \alpha, \beta)$ and the witness $\mathbb{w} = (b_1, \dots, b_n) \in \mathbb{F}^n$ as follows.

$\mathcal{R}_{\text{GraphStatus}}(\mathbb{x} ; \mathbb{w}) = 1$ iff the following constraints hold:

$$\text{cm} = \sum_{i=1}^n b_i [L_i(\tau)]_1 \tag{1}$$

$$\forall i \in V : b_i(b_i - 1) = 0 \tag{2}$$

$$\sum_{i=1}^n b_i \geq \alpha \tag{3}$$

$$\forall i \in V : b_i \left(\sum_{j \in \mathcal{N}(i)} b_j \right) \geq b_i \beta \tag{4}$$

Remark 3 (SNARK Encoding & Complexity). We formalize a relation proving knowledge of a vertex subset of size at least α satisfying a local degree threshold of at least β . To compile this into a R1CS (e.g., Groth16), linear combinations (such as summing neighbors) are practically free. The booleanity constraint (2) requires exactly n constraints. The inequality constraints (3) and (4) are efficiently encoded using slack variables and bit-decomposition range proofs. Since the neighborhood size is sparse ($|\mathcal{N}(i)| = O(\log n)$), these local range proofs require only $O(\log \log n)$ constraints per client. Combined with the $O(n)$ constraints needed to encode the commitment (1) inside the circuit, the overall prover complexity scales at $O(n \log \log n)$. This lightweight circuit preserves an $O(1)$ verifier time and proof size (e.g., 3 group elements).