Alg{Ossa}: Decentralized Secure Aggregation with Rotating Committees and Eventual Delivery

Xiangyu Su[1], Yuma Tamagawa[1], Mario Larangeira[1,2], Zengpeng Li[3], and Keisuke Tanaka[1]

[1] Institute of Science Tokyo. su.x.4029@m, tamagawa.y.f6c3@m, rebello.m.f72a@m, keisuke@comp.isct.ac.jp.
[2] Input Output, Singapore. mario.larangeira@iohk.io.
[3] Shandong University. zengpengliz@gmail.com.

Abstract. Secure aggregation lets clients reveal only an aggregate of their private contributions, supporting a variety of applications, including privacy-preserving sensing, collaborative analytics, and federated learning. Building on the single-server, committee-aided model for one-shot clients used by OPA and Willow (CRYPTO '25), we present *Ossa*, a generic construction with a modular security analysis. Ossa realizes the standard secure aggregation functionality with non-selective abort against a malicious server, but leaves role selection unspecified and provides no delivery guarantee. We therefore present *AlgOssa*, which integrates Ossa into an Algorand-style consensus backbone in which cryptographic sortition selects fresh potential leaders and verifier committees in each round and refreshes the decryptor committee from finalized chain state at key epochs. AlgOssa further selects the server from among the potential leaders, and we establish an explicit lower bound on the probability that the selected server is honest. Consequently, the probability that a round delivers a non-empty aggregate is bounded below by a positive constant, yielding an *eventual delivery guarantee* across repeated rounds. This variable delivery cadence fits delay-tolerant statistical sensing and, with persistent buffering and staleness control, buffered asynchronous federated learning. Our generic core supports classical and post-quantum instantiation routes; we evaluate complete local classical rounds and demonstrate the feasibility of the post-quantum route.

1 Introduction

Secure aggregation allows clients to contribute private values while revealing only their aggregate. Although frequently motivated by federated learning, it also supports privacy-preserving telemetry and aggregate analytics [26], as well as repeated aggregation in sensor and IoT deployments [39].

A substantial line of work studies secure aggregation in the single-server setting, using reusable client state or committee assistance to reduce its cost. Flamingo [46] amortizes setup over repeated aggregations and periodically rotates its decryptors, whereas LERNA [39] uses a fixed, stateful committee to

hold shares of reusable client masking keys. OPA [37] and Willow [8] shift the emphasis toward one-shot clients (*i.e.*, one message per aggregation): OPA uses ephemeral, stateless committees, while Willow supports dynamic client participation through committee-aided verification and decryption. WillowFold [36] subsequently reduces the committee's verification cost using proof-carrying data. These protocols progressively move state, interaction, and cryptographic work away from ordinary clients and into auxiliary committees. However, the system boundary remains largely unchanged: a designated server coordinates the aggregation, while committee membership is fixed during setup or supplied by an external selection mechanism. Consequently, changing the committee between aggregations does not itself replace an unavailable server, establish agreement on which server should act, or determine a canonical aggregation proposal. OPA, for example, provides malicious security with non-selective abort. Willow additionally guarantees output delivery to an honest or semi-honest server under its committee assumptions, through a strengthened setup and additional communication.

Our work explores a complementary system-level approach for deployments with consensus infrastructure. We ask: *how can we retain the efficiency of a logically single aggregation server without relying on a fixed, potentially malicious server for progress?*

1.1 Contributions and Technical Overview

We propose AlgOssa, a decentralized secure aggregation protocol with rotating committees and eventual output delivery. Its design separates the secure aggregation core from the mechanisms responsible for role assignment and protocol progress. We describe the two protocol layers and our implementation below; applications and deployment requirements are discussed in Appendix G.

Ossa: a modular secure aggregation core (Sec. 3). Ossa realizes the secure aggregation functionality with non-selective abort against a malicious server, as considered in Willow [8]. Its construction combines the following design choices.

- The aggregation follows the additively homomorphic encryption (AHE) approach used in [55], later extended with secret sharing-based decryptor dropout recovery and deployed as the core of Willow [8]. We instead use a generic threshold AHE (TAHE) interface with explicit correctness and security requirements, abstracting how threshold decryption is implemented.
- Since applying TAHE directly to high-dimensional client vectors is inefficient, we adopt seed-homomorphic pseudorandom generator (SHPRG)-based masking [40] to confine the expensive encryption work to compact seeds. Willow's PRG and key-additive homomorphic symmetric encryption (KAHE) pipeline [8] and OPA's key-homomorphic pseudorandom function variant [37] serve a similar purpose: reducing the effective input dimension handled by the threshold encryption or secret sharing. We combine SHPRG masking with the generic TAHE interface to recover the sum of encrypted seeds. Fresh per-round seeds avoid long-lived client masking state, matching the decentralized,

role-rotated setting of AlgOssa, where any required coordination state can be obtained from the consensus state.
– Finally, we use a committee-aided verification and decryption structure, in which verifiers certify aggregation validity before decryptors release partial decryption shares. Since security against a malicious server requires the simulator to generate valid ciphertexts for honest clients before the aggregation set is fixed, we incorporate an additional hash-based masking term, with the hash modeled as a programmable random oracle as in OPA [37]. An auxiliary threshold public key encryption (TPKE) mechanism enables reconstruction and subtraction of the aggregate hash mask. For fixed committee sizes, this preserves the asymptotic dependence on the number of clients compared with Willow [8], whose PRG seeds are likewise protected by auxiliary encryption for malicious server simulation.

We treat the SHPRG, TAHE, TPKE, and submission argument as abstract building blocks and prove security from their required properties. This modular construction supports classical and post-quantum (PQ) instantiations (Appendix D). Separate client, server, verifier, and decryptor interfaces let an external coordination layer reassign roles without changing the aggregation protocol.

From Ossa to AlgOssa (Sec. 4). The standalone Ossa protocol leaves two system-level questions open: who executes the server and committee interfaces, and how delivery proceeds when the designated server refuses or fails to act.

AlgOssa addresses these questions through a decentralized wrapper around Ossa. Algorand-style cryptographic sortition [22,11] selects potential leaders and fresh verifier committees in each round, while the finalized chain state determines a refreshed decryptor committee at each key epoch. Potential leaders construct aggregation proposals, and consensus finalizes either one eligible proposal or the canonical empty block. Decryptors release partial decryptions only for a finalized aggregation payload, binding reconstruction to the agreed outcome. It retains Ossa's tolerance of arbitrary client corruption and assumes an honest super-majority $f > 2/3$ among consensus nodes, yielding the required committee quorum properties except with negligible probability. Leader selection gives an explicit positive constant lower bound on the probability of an honest selected server; together with consensus termination and Ossa's participation condition, this yields a corresponding bound on non-empty aggregate delivery in each round. Individual rounds may finalize the canonical empty block; repeated rounds provide an *eventual delivery guarantee*.

Implementation and evaluation (Sec. 5 and Appendix F). We implement classical and PQ instantiations of the generic Ossa core. We evaluate complete local classical rounds, reporting arithmetic preparation, per-role computation, and payload sizes, including the implemented proofs and verification checks. We also measure scalability with client count and vector length and assess PQ feasibility. For context, we include measurements from our extension of Willow's public code with auxiliary masking and key recovery, and from our OPA implementation, with security and implementation coverage differences stated explicitly.

1.2 Related Work

We survey the secure aggregation literature. A concise comparison of the most relevant protocols appears in Tab. 3 of Appendix A.

Single-server secure aggregation. The modern line of single-server secure aggregation for federated learning begins with [12], which uses pairwise masks that cancel in the aggregate. Establishing these masks and recovering from client dropouts require multiple rounds of coordinated client communication. Later work [7] reduces this overhead by replacing the complete pairwise communication graph with a logarithmic-degree graph. SASH [40] instead uses an SHPRG to aggregate short seeds while masking high-dimensional inputs. Other single-server systems, including EIFFeL [25], ACORN [6], and Armadillo [45], additionally incorporate input validation for hidden client contributions. A complementary line shifts state, interaction, and cryptographic work from ordinary clients to auxiliary committees [46,39,37,8,36], while retaining a designated aggregation server.

Multi-server secure aggregation. The Prio line [26,3,28] distributes secret-shared inputs among a small set of computation servers, while PINE [59] provides efficient norm validation for such inputs. ELSA [57], Whisper [58], and SealAgg [32] explore multi-server designs for privacy against malicious servers, efficiency, and one-shot aggregate auditing, respectively. SCIF [61] additionally provides guaranteed output delivery and input inclusion against fewer than one-third malicious computation servers.

Functional encryption-based aggregation. At the primitive level, multi-client functional encryption and its variants with decentralized functional key generation [24,2,1,51] enable private linear aggregation by evaluating inner products over independently encrypted client inputs.

Consensus-assisted aggregation. Blockchain-assisted federated learning systems couple private or robust aggregation with consensus. PBFL [48] combines fully homomorphic encryption-based aggregation and update filtering with blockchain transparency. BlockDFL [56] selects aggregators and verifiers using the preceding block hash and participant stake, and uses PBFT-style voting to select a global update. FLock [23] combines poisoning-robust aggregation with a multi-party state channel coordinated by BFT.

These works do not establish secure aggregation against malicious aggregators: PBFL assumes noncolluding honest-but-curious servers, FLock assumes semi-honest MPC aggregators, and BlockDFL relies on empirical resistance to gradient reconstruction. Their consensus analyses also leave progress under faulty protocol roles insufficiently justified: PBFL relies on blockchain logging and incentives, BlockDFL informally argues the properties of its modified PBFT voting, and FLock's liveness proof considers only normal operation with a stable leader. In contrast, AlgOssa separates the two security foundations: Ossa is analyzed against the secure aggregation functionality under malicious corruption, while its Algorand-style outer layer establishes role selection, agreement, and progress.

2 Preliminaries

We use $\lambda \in \mathbb{N}, \lambda > 0$ for the security parameter. $\mathsf{poly}(\cdot)$ and $\mathsf{negl}(\cdot)$ denote a polynomial and a negligible function, respectively. PPT is short for probabilistic polynomial time. We use ":=" for deterministic assignment, "$\leftarrow$" for assignment to an algorithm's output, and "$\leftarrow\!\!\$$" for uniform random sampling. We use $\mathbf{u}$ to denote a vector whose entries are sampled independently and uniformly from a given domain. The integer range $[a \,..\, b]$ represents $\{a, a+1, \ldots, b\}$, $[n]$ denotes $[1 \,..\, n]$, and $[n]_0$ denotes $[0 \,..\, n]$.

We recall the syntax of SHPRG, TPKE, TAHE, and non-interactive arguments. Additionally, we recall a signature scheme $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$ in Appendix B.1.

SHPRG. A PRG is an efficiently computable function $\mathsf{PRG} : \mathcal{K} \to \mathcal{Y}$, parameterized by a pair of algorithms $\mathsf{PRG} = (\mathsf{Setup}, \mathsf{Expand})$, where $(\mathcal{K}, \oplus)$ and $(\mathcal{Y}, \otimes)$ are groups:

- $\mathsf{Setup}(1^\lambda)$ takes as input the security parameter λ and outputs the public parameters pp.
- $\mathsf{Expand}(\mathsf{pp}, \mathsf{sd})$ takes as input pp and a seed $\mathsf{sd} \in \mathcal{K}$. It outputs $Y = \mathsf{PRG}(\mathsf{sd}) \in \mathcal{Y}$ *s.t.* $|Y| > |\mathsf{sd}|$.

This work employs an SHPRG scheme that is secure under leakage of the sum of seeds [37].[1]

Definition 1 (Leakage-Resilient SHPRG). $(\mathsf{PRG}, \oplus, \otimes)$ *is a secure SHPRG under leakage of sum of seeds if:*

- PRG *is a secure PRG under leakage of sum of seeds, i.e., for any* PPT *adversary* $\mathcal{A}$*, the following distribution ensembles are computationally indistinguishable, denoted by* $\mathcal{D}_0 \overset{c}{\approx} \mathcal{D}_1$*, for any* λ*,* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, and* $\mathsf{sd}, \mathsf{sd}' \leftarrow\!\!\$\ \mathcal{K}$*.*
$$\mathcal{D}_0 := \left\{ \left(Y_0 := \mathsf{Expand}(\mathsf{pp}, \mathsf{sd}), \mathsf{sd} \oplus \mathsf{sd}' \right) \right\},$$
$$\mathcal{D}_1 := \left\{ \left(Y_1 \leftarrow\!\!\$\ \mathcal{Y}, \mathsf{sd} \oplus \mathsf{sd}' \right) \right\}$$
- *For any* λ*,* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, and* $\mathsf{sd}_1, \mathsf{sd}_2 \in \mathcal{K}$*,* $\mathsf{Expand}(\mathsf{pp}, \mathsf{sd}_1) \otimes \mathsf{Expand}(\mathsf{pp}, \mathsf{sd}_2) = \mathsf{Expand}(\mathsf{pp}, \mathsf{sd}_1 \oplus \mathsf{sd}_2)$.

TPKE and TAHE. A TPKE scheme consists of a tuple of algorithms $\mathsf{PKE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{PartDec}, \mathsf{PartVf}, \mathsf{Combine})$. Specifically, we consider the standard t-out-of-n setting.

- $\mathsf{Setup}(1^\lambda, n, t)$ takes as input the security parameter λ, the number of parties n, and a threshold value $t \in [n]$. It outputs public parameters pp that specifies the message space $\mathcal{M}$ and the ciphertext space $\mathcal{C}$, an encryption key ek, and a set of decryption keys $\{dk_i\}_{i \in [n]}$ (one for each party). All subsequent procedures implicitly take pp as input.

[1] For ease of exposition, we state exact seed homomorphism in Def. 1 and the protocol descriptions. For the LWR-based *almost seed-homomorphic* instantiation of [15], we have $\mathsf{Expand}(\mathsf{pp}, \mathsf{sd}_1) \otimes \mathsf{Expand}(\mathsf{pp}, \mathsf{sd}_2) = \mathsf{Expand}(\mathsf{pp}, \mathsf{sd}_1 \oplus \mathsf{sd}_2) \otimes e$, where e is a bounded rounding error. We handle the accumulated error through input encoding and decoding as in [37].

– $\mathsf{Enc}(ek, m)$ takes as input an encryption key ek and a plaintext $m \in \mathcal{M}$. It outputs a ciphertext $ct \in \mathcal{C}$.
– $\mathsf{PartDec}(dk_i, ct)$ takes as input a decryption key share dk_i and a ciphertext ct. It outputs a partial decryption share d_i.
– $\mathsf{PartVf}(ek, ct, d_i)$ takes as input an encryption key ek, a ciphertext ct, and a partial decryption share d_i. It outputs 1 if d_i is valid, or 0 otherwise.
– $\mathsf{Combine}(ek, ct, \{\phi(d_i)\}_{i\in S})$ takes as input an encryption key ek, a ciphertext ct, and a set of images of ϕ of partial decryption shares $\{\phi(d_i)\}_{i\in S}$. It outputs a plaintext $m' \in \mathcal{M} \cup \{\perp\}$. The function ϕ is public and deterministic.[2]

We extend the syntax with an $\mathsf{EvalSum}$ algorithm to support additive homomorphism, yielding a TAHE scheme, denoted by $\mathsf{AHE} = \mathsf{PKE} \cup \{\mathsf{EvalSum}\}$. The homomorphism is given by decryption correctness below (Def. 4).

– $\mathsf{EvalSum}(ek, \{ct_i\}_{i\in[k]})$ takes as input an encryption key ek and a set of ciphertexts $\{ct_i\}_{i\in[k]}$. It outputs a ciphertext $\hat{ct}$.

Remark 1 (Centralized setup in protocol descriptions). For ease of exposition, the descriptions of Ossa (Sec. 3) and AlgOssa (Sec. 4) model $\mathsf{PKE.Setup}$ and $\mathsf{AHE.Setup}$ as modular *centralized* procedures. In the concrete instantiation (Appendix D), we replace them with robust distributed key generation (DKG) protocols.

We require both TPKE and TAHE to satisfy decryption correctness, partial verification correctness, threshold IND-CPA security, and robustness; TAHE must additionally satisfy simulation security. The formal definitions are given in Sec. B.2.

Non-interactive arguments of knowledge Let $\mathcal{R} \subseteq \{0,1\}^* \times \{0,1\}^*$ be an NP relation, parameterized by the security parameter λ. Denote the language defined by $\mathcal{R}$ as $L_{\mathcal{R}} := \{x \in \{0,1\}^* | \exists w \in \{0,1\}^{\mathsf{poly}(|x|)} \; s.t. \; (x,w) \in \mathcal{R}\}$. A non-interactive argument protocol for $\mathcal{R}$ consists of a tuple of algorithms $\Pi_{\mathcal{R}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Vf})$.

– $\mathsf{Setup}(1^\lambda, \mathcal{R})$ takes as input λ and a relation $\mathcal{R}$. It outputs a common reference string crs and a trapdoor td.
– $\mathsf{Prove}(\mathsf{crs}, x, w)$ takes as input crs, an instance x, and a witness w. It outputs an argument π.
– $\mathsf{Vf}(\mathsf{crs}, x, \pi)$ takes as input crs, an instance x, and an argument π. It outputs 1 if π is valid, or 0 otherwise.

We require a NIZKAoK that satisfies completeness, zero-knowledge, knowledge soundness, and simulation extractability (SE); formal definitions appear in Sec. B.3.

3 Ossa: The Secure Aggregation Core

This section presents Ossa, a one-shot secure aggregation protocol that constitutes the core of our decentralized protocol AlgOssa. We begin by detailing the

[2] As noted in [29, Footnote 5], ϕ-projection is for defining robustness. In protocol descriptions, we implicitly apply ϕ to each share passed to $\mathsf{Combine}$.

setting and presenting a secure aggregation functionality $\mathcal{F}_{\text{Agg}}$. We then give a modular and generic construction Π_{SA}, together with a simulation-based proof that Π_{SA} securely realizes $\mathcal{F}_{\text{Agg}}$ in the random oracle model (ROM).

3.1 Setting and Functionality

The Ossa protocol follows the conceptual role structure of [8]: client, server, (aggregation) verifier, and decryptor. We target a single-server setting with one-shot clients (*i.e.*, each client sends one message to the server per aggregation round) and committee-based verifier and decryptor roles.

- Clients $\{\mathsf{C}_i\}_{i\in[n_{\text{c}}]}$ are the providers of private inputs.
- Server S collects client submissions and, with the assistance of the verifier and decryptor committees, outputs the aggregation result of the private inputs.
- Verifier committee $\{\mathsf{V}_i\}_{i\in[n_{\text{v}}]}$ verifies a transcript produced by the server *w.r.t.* a public aggregation validity predicate.
- Decryptor committee $\{\mathsf{D}_i\}_{i\in[n_{\text{d}}]}$ releases decryption shares that enable recovery of the aggregation result.

Threat model and assumptions. We consider the security against a static malicious adversary $\mathcal{A}$. That is, $\mathcal{A}$ fixes the set of corrupted parties (including the single server) before protocol execution; upon corrupting a party, $\mathcal{A}$ learns its entire secret state and thereafter fully controls its actions. We denote the fraction of honest clients by f_{c}, honest verifier committee members by f_{v}, and honest decryptor committee members by f_{d}. All corrupted parties may collude under $\mathcal{A}$'s coordination. Moreover, we assume an external Public Key Infrastructure (PKI) (or a verifiable public key directory [47]) that allows parties to agree on each other's public keys and thus establish authenticated point-to-point channels. These assumptions are chosen to accommodate our decentralized protocol AlgOssa; we motivate them as follows.

The corruption model. We adopt the static corruption model since it matches the standard threat model used for the underlying TPKE and TAHE (cf. Appendix B.2). While static corruption is less realistic for multi-round decentralized protocols, it is adopted by most state-of-the-art standalone secure aggregation protocols, *e.g.*, [46,37,8]. For AlgOssa, a possible extension for this assumption is a *slowly-adaptive* model [64], in which the corrupted set may evolve over time but remains fixed within each epoch (consisting of multiple rounds). Supporting this relaxation typically requires epoch-wise refresh of the threshold key materials, *e.g.*, by rerunning the (distributed) setup of TPKE/TAHE at epoch boundaries.

The honest fractions. At the Ossa level, we distinguish between the honest fractions of clients, verifiers, and decryptors. Similar to [37,8], Ossa tolerates an arbitrary client corruption rate, while the verifier and decryptor committees are assumed to contain an honest super-majority, *i.e.*, $f_{\text{v}}, f_{\text{d}} > 2/3$. For AlgOssa, decentralization is realized by selecting the potential leader (executing the server interface of Ossa), verifier, and decryptor roles from a common population of consensus nodes. We additionally allow the client population and consensus nodes to be distinct, which is useful when clients are lightweight devices that only submit

inputs. Accordingly, we assume the adversary controls at most a $1 - f$ fraction of the consensus nodes. Our analysis focuses on the standard committee-based consensus regime, *i.e.*, $f > 2/3$. The verifier and decryptor committees are sampled with parameters chosen to keep their honest fractions close to that of the underlying population, preserving honest super-majorities except with negligible probability. We establish the corresponding concentration bounds and quorum guarantees in Sec. 4.2 (cf. Eq. 4.1).

The PKI assumption. AlgOssa requires a global PKI, as is standard in committee-based consensus protocols [22,11]. In Ossa, we use this PKI to authenticate clients and verifiers. Verifiers check client signatures to prevent server-side substitution, ensuring that the honest client indices counted in Lemma 1 correspond to those clients' actual submissions. Willow [8] deliberately avoids client PKI and notes that Sybil attacks can defeat the protection provided by a minimum aggregation size. OPA omits client signatures from its main protocol presentation but explicitly discusses them [37, Remark 2], together with a client PKI, as protection against the same substitution attack.

Secure aggregation functionality. We recall the aggregation functionality with non-selective abort of [8] and present it as $\mathcal{F}_{\text{Agg}}$ (Fig. 1) in an interface-based form. We further discuss its features.

The $n_{\min}$ requirement w.r.t. honest client fraction f_c. Unlike [8], which treats $n_{\min}$ as an application-level parameter, we relate $n_{\min}$ to the honest fraction f_c. Since $\mathcal{F}_{\text{Agg}}$ reveals only the sum, we choose:

$$n_{\min} \geq (1 - f_c)n_c + 2 \tag{3.1}$$

to rule out an aggregate containing only one honest input. The following lemma formalizes this guarantee; its proof follows directly from the bound on the number of corrupted clients.

Lemma 1. *Let n_c be the number of clients and f_c the honest fraction. For any client index set S with $|S| \geq n_{\min}$, where $n_{\min}$ satisfies Eq. 3.1, S contains at least two honest client indices.*

Correctness with non-selective abort. The functionality $\mathcal{F}_{\text{Agg}}$ permits only non-selective abort: the adversary may force the protocol to output $\perp$ without first learning the aggregate. Otherwise, if an output is produced, it must equal the correct aggregation result for the selected set S^*, and the adversary can influence it only through the inputs of corrupted clients. We do not aim to guarantee output delivery at the standalone secure aggregation layer (cf. [8, Section 6.7]). Instead, we treat delivery in AlgOssa: leveraging the liveness of the decentralized backbone, we obtain an eventual delivery guarantee via probabilistic analysis.

Input validation is left outside of $\mathcal{F}_{\text{Agg}}$ and can be incorporated modularly using existing mechanisms, *e.g.*, [6,41].

Functionality $\mathcal{F}_{\mathrm{Agg}}(\mathbb{F}^{\ell}, n_{\mathrm{c}}, n_{\min})$

Public Parameters: The input space $\mathbb{F}^{\ell}$, a finite abelian group written additively; the number of clients n_{c}; and the minimum aggregation batch size $n_{\min} \in [n_{\mathrm{c}}]$.

Parties: Server S; n_{c} clients $\{\mathsf{C}_i\}_{i\in[n_{\mathrm{c}}]}$, each holding a private input[a] $\mathbf{x}_i \in \mathbb{F}^{\ell}$; and the adversary $\mathcal{A}$ corrupting a subset of parties in $\{\mathsf{C}_i\}_{i\in[n_{\mathrm{c}}]} \cup \{\mathsf{S}\}$.

State: Client submission list $\mathsf{L}_{\mathrm{inp}} := (\bot, \ldots, \bot)$ of size n_{c}; finalization state $\mathsf{closed} := 0$; and the finalized aggregation set $S^* := \bot$.

Submit. Upon receiving $(\mathrm{submit}, i, \mathbf{x}_i)$ from C_i:[b]

1. If $\mathsf{closed} = 1$ or $\mathsf{L}_{\mathrm{inp}}[i] \neq \bot$, ignore.
2. Set $\mathsf{L}_{\mathrm{inp}}[i] := \mathbf{x}_i$.
3. Send (submit, i) to S and to $\mathcal{A}$.

Finalize. Upon receiving $(\mathrm{finalize}, S)$ from S:

1. If $\mathsf{closed} = 1$, ignore.
2. Set $\mathsf{closed} \leftarrow 1$.
3. Define $S^{\mathrm{all}} := \{i \in [n_{\mathrm{c}}] : \mathsf{L}_{\mathrm{inp}}[i] \neq \bot\}$.
4. If S is honest, set $S^* := S^{\mathrm{all}}$.
5. If S is corrupted, set $S^* := S$.
6. Send (chosen, S^*) to $\mathcal{A}$.

Deliver. For $d \in \{\mathrm{continue}, \mathrm{abort}\}$, upon receiving $(\mathrm{deliver}, d)$ from $\mathcal{A}$:

1. If $d = \mathrm{abort}$, send $\bot$ to S and halt.
2. If $\mathsf{closed} = 0$, send $\bot$ to S and halt.
3. If $|S^*| < n_{\min} \vee S^* \not\subseteq S^{\mathrm{all}}$, send $\bot$ to S and halt.
4. Send $y \leftarrow \sum_{i\in S^*} \mathsf{L}_{\mathrm{inp}}[i]$ to S and to $\mathcal{A}$ and halt.

[a] The inputs of corrupted clients are chosen by $\mathcal{A}$.

[b] The interface models successful submissions; those lost in transit are not recorded.

Fig. 1: Non-selective abort secure aggregation functionality.

3.2 Our Ossa Construction

We present our Ossa construction as a tuple of algorithms:

$$\Pi_{\mathsf{SA}} = (\mathsf{Setup}, \mathsf{Submit}, \mathsf{Aggregate}, \mathsf{Vf}, \mathsf{Dec}, \mathsf{Deliver}).^{3}$$

The construction uses the following building blocks: a hash function $\mathsf{H} : \mathcal{D}_{\mathsf{H}} \to \mathbb{F}^{\ell}$, a signature scheme SIG, an SHPRG PRG, a TPKE scheme PKE, a TAHE scheme AHE, and a NIZKAoK protocol $\Pi_{\mathcal{R}_{\mathrm{val}}}$ for a submission well-formedness relation $\mathcal{R}_{\mathrm{val}}$, which is specified under $\Pi_{\mathsf{SA}}.\mathsf{Submit}$ (cf. Eq. 3.2).

Protocol setup. We describe $\Pi_{\mathsf{SA}}.\mathsf{Setup}$ (Fig. 2) as a centralized setup procedure for conceptual simplicity (cf. Remark 1). It takes as input the security parameter λ, the protocol participation configuration $\mathsf{cfg} = (n_{\mathrm{c}}, n_{\mathrm{v}}, n_{\mathrm{d}}, n_{\min})$, the submission well-formedness relation $\mathcal{R}_{\mathrm{val}}$, a certification threshold t_{v} (determined as a function of n_{v} and f_{v} and specified in $\Pi_{\mathsf{SA}}.\mathsf{Vf}$), and a decryption threshold t_{d} (determined as a function of n_{d} and f_{d} and specified in $\Pi_{\mathsf{SA}}.\mathsf{Deliver}$). It produces the public parameters pp by running the setup algorithms of PRG,

[3] In Fig. 2 to 7, blue content highlights the hash-based masking mechanism used to enable simulation against a malicious server.

PKE, AHE, and $\Pi_{\mathcal{R}_{\mathrm{val}}}$. Notably, PKE.Setup and AHE.Setup are parameterized by n_{d} and t_{d}, and $\Pi_{\mathcal{R}_{\mathrm{val}}}$.Setup is parameterized by $\mathcal{R}_{\mathrm{val}}$. We also treat signature key generation for clients and verifiers as part of the setup procedure: each client and verifier generates its own key pair locally and registers the public key in the PKI.

$\Pi_{\mathsf{SA}}.\mathsf{Setup}(1^\lambda, \mathsf{cfg}, t_{\mathrm{v}}, t_{\mathrm{d}}, \mathcal{R}_{\mathrm{val}})$

1. Parse $\mathsf{cfg} = (n_{\mathrm{c}}, n_{\mathrm{v}}, n_{\mathrm{d}}, n_{\min})$.
2. Execute $\mathsf{pp}_{\mathsf{PRG}} \leftarrow \mathsf{PRG.Setup}(1^\lambda)$,
   $(\mathsf{pp}_{\mathsf{PKE}}, ek_{\mathsf{PKE}}, \{dk_{\mathsf{PKE},i}\}_{i\in[n_{\mathrm{d}}]}) \leftarrow \mathsf{PKE.Setup}(1^\lambda, n_{\mathrm{d}}, t_{\mathrm{d}})$,
   $(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \{dk_{\mathsf{AHE},i}\}_{i\in[n_{\mathrm{d}}]}) \leftarrow \mathsf{AHE.Setup}(1^\lambda, n_{\mathrm{d}}, t_{\mathrm{d}})$,
   $(\mathsf{crs}, \mathsf{td}) \leftarrow \Pi_{\mathcal{R}_{\mathrm{val}}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathrm{val}})$.
3. For each client $i \in [n_{\mathrm{c}}]$ and verifier $j \in [n_{\mathrm{v}}]$, obtain $pk_{\mathrm{c},i}$ and $pk_{\mathrm{v},j}$, where SIG.KGen is run locally.
4. Set public parameters:
   $$\mathsf{pp} := (\mathsf{cfg}, t_{\mathrm{v}}, t_{\mathrm{d}}, \mathsf{pp}_{\mathsf{PRG}}, \mathsf{pp}_{\mathsf{PKE}}, \mathsf{pp}_{\mathsf{AHE}}, \mathsf{crs}).$$
   Send pp to all participants of Π_{SA}.
5. Register $(ek_{\mathsf{PKE}}, ek_{\mathsf{AHE}}, \{pk_{\mathrm{c},i}\}_{i\in[n_{\mathrm{c}}]}, \{pk_{\mathrm{v},j}\}_{j\in[n_{\mathrm{v}}]})$ in the PKI. For each $i \in [n_{\mathrm{d}}]$, send $(dk_{\mathsf{PKE},i}, dk_{\mathsf{AHE},i})$ privately to the decryptor committee member indexed by i.

Fig. 2: The *conceptual* centralized setup procedure.

Submission of masked inputs. $\Pi_{\mathsf{SA}}.\mathsf{Submit}$ (Fig. 3) first masks the client C_i's input using an SHPRG seed sd_i via PRG.Expand and encrypts sd_i via AHE.Enc. To facilitate simulation against a malicious server, the client additionally adds a hash-based mask $\mathsf{H}(\mathsf{dig}_i)$, and encrypts dig_i via PKE.Enc. It then proves submission well-formedness and signs the submission.

We treat the client's registered public key $pk_{\mathrm{c},i}$ as its identifier. All interfaces of Π_{SA} share an implicit execution identifier, instantiated by the epoch and round indices (e, r) in AlgOssa. The $\Pi_{\mathsf{SA}}.\mathsf{Aggregate}$, $\Pi_{\mathsf{SA}}.\mathsf{Vf}$, and $\Pi_{\mathsf{SA}}.\mathsf{Dec}$ interfaces reject proofs and signatures not bound to this identifier. For any instance $x_i = (pk_{\mathrm{c},i}, \mathbf{y}_i, ct_{\mathsf{sd}_i}, ct_{\mathsf{dig}_i})$ and witness $w_i = (\mathbf{x}_i, \mathsf{sd}_i, r_{\mathsf{sd}_i}, \mathsf{dig}_i, r_{\mathsf{dig}_i})$, we define the well-formedness relation $\mathcal{R}_{\mathrm{val}}(x_i, w_i) = 1$ iff:

$$\begin{aligned} &\mathbf{y}_i = \mathbf{x}_i + \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i) + \mathsf{H}(\mathsf{dig}_i) \\ \wedge\, &ct_{\mathsf{sd}_i} = \mathsf{AHE.Enc}(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \mathsf{sd}_i; r_{\mathsf{sd}_i}) \\ \wedge\, &ct_{\mathsf{dig}_i} = \mathsf{PKE.Enc}(\mathsf{pp}_{\mathsf{PKE}}, ek_{\mathsf{PKE}}, \mathsf{dig}_i; r_{\mathsf{dig}_i}). \end{aligned} \tag{3.2}$$

Aggregation of masked inputs. Run by S, $\Pi_{\mathsf{SA}}.\mathsf{Aggregate}$ (Fig. 4) first verifies π_i and $\sigma_{\mathrm{c},i}$ for each received submission. The resulting set S^{val} is local to the server and need not be consistent across the network, *e.g.*, due to lost client submissions.

An honest server aborts if $|S^{\mathrm{val}}| < n_{\min}$ and otherwise sets $S := S^{\mathrm{val}}$. A malicious server may instead propose any subset of valid submissions $S \subseteq S^{\mathrm{val}}$ satisfying $|S| \geq n_{\min}$ (cf. Eq. 3.1); these conditions are enforced by the verifier committee. By Lemma 1, this ensures that S contains at least two honest client indices whenever an aggregate is produced. The server forms an *aggregation*

$\Pi_{\mathsf{SA}}.\mathsf{Submit}(\mathsf{pp}, sk_{\mathrm{c},i}, ek_{\mathsf{PKE}}, ek_{\mathsf{AHE}}, \mathbf{x}_i)$[a]

1. Parse the SHPRG key space $\mathcal{K}$ from $\mathsf{pp}_{\mathsf{PRG}} \in \mathsf{pp}$.
2. Sample a seed $\mathsf{sd}_i \leftarrow\!\!\$\ \mathcal{K}$ and a hash seed $\mathsf{dig}_i \leftarrow\!\!\$\ \mathcal{D}_{\mathsf{H}}$. Compute the masked input:
$$\mathbf{y}_i := \mathbf{x}_i + \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i) + \mathsf{H}(\mathsf{dig}_i)$$
3. Encrypt sd_i using fresh randomness r_{sd_i} via $\mathsf{AHE.Enc}$:
$$ct_{\mathsf{sd}_i} := \mathsf{AHE.Enc}(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \mathsf{sd}_i; r_{\mathsf{sd}_i}).$$
4. Encrypt dig_i using fresh randomness r_{dig_i} via $\mathsf{PKE.Enc}$:
$$ct_{\mathsf{dig}_i} := \mathsf{PKE.Enc}(\mathsf{pp}_{\mathsf{PKE}}, ek_{\mathsf{PKE}}, \mathsf{dig}_i; r_{\mathsf{dig}_i}).$$
5. Set the instance $x_i := (pk_{\mathrm{c},i}, \mathbf{y}_i, ct_{\mathsf{sd}_i}, ct_{\mathsf{dig}_i})$ and witness $w_i := (\mathbf{x}_i, \mathsf{sd}_i, r_{\mathsf{sd}_i}, \mathsf{dig}_i, r_{\mathsf{dig}_i})$. Generate the proof:
$$\pi_i \leftarrow \Pi_{\mathcal{R}_{\mathrm{val}}}.\mathsf{Prove}(\mathsf{crs}, x_i, w_i).$$
6. Sign the instance and proof with $\sigma_{\mathrm{c},i} \leftarrow \mathsf{SIG.Sign}(sk_{\mathrm{c},i}, (x_i, \pi_i))$. Send $(x_i, \pi_i, \sigma_{\mathrm{c},i})$ to the server S.

[a]Throughout, the executing party's public key is implicit in interface inputs.

Fig. 3: The submission interface for client C_i, $\forall i \in [n_{\mathrm{c}}]$.

message $m = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_i}\}_{i\in S})$ containing the decryption data and sends it with the *transcript* $\{(pk_{\mathrm{c},i}, \mathbf{y}_i, ct_{\mathsf{sd}_i}, \pi_i, \sigma_{\mathrm{c},i})\}_{i\in S}$ to the verifier committee for validation.

$\Pi_{\mathsf{SA}}.\mathsf{Aggregate}(\mathsf{pp}, \{(x_i, \pi_i, \sigma_{\mathrm{c},i})\}_{i\in[n_{\mathrm{c}}]})$

1. Parse $x_i = (pk_{\mathrm{c},i}, \mathbf{y}_i, ct_{\mathsf{sd}_i}, ct_{\mathsf{dig}_i})$.
2. Define the index set of clients with valid submissions:[a]
$$S^{\mathrm{val}} := \left\{ i \in [n_{\mathrm{c}}] : \begin{array}{c} \Pi_{\mathcal{R}_{\mathrm{val}}}.\mathsf{Vf}(\mathsf{crs}, x_i, \pi_i) = 1 \\ \wedge \mathsf{SIG.Vf}(pk_{\mathrm{c},i}, (x_i, \pi_i), \sigma_{\mathrm{c},i}) = 1 \end{array} \right\},$$
and abort with $\perp$ if $|S^{\mathrm{val}}| < n_{\min}$.
3. Set an aggregation set $S := S^{\mathrm{val}}$. Compute: $\mathbf{y} = \sum_{i\in S} \mathbf{y}_i$,
$$ct_{\mathsf{sd}} = \mathsf{AHE.EvalSum}(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \{ct_{\mathsf{sd}_i}\}_{i\in S}).$$
4. Define an aggregation message:
$$m := (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_i}\}_{i\in S}),$$
and broadcast $(m, \{(pk_{\mathrm{c},i}, \mathbf{y}_i, ct_{\mathsf{sd}_i}, \pi_i, \sigma_{\mathrm{c},i})\}_{i\in S})$ to the verifier committee $\{\mathsf{V}_j\}_{j\in[n_{\mathrm{v}}]}$.[b]

[a]All interfaces discard messages with unregistered sender public keys.
[b]$\{ct_{\mathsf{sd}_i}\}_{i\in S}$ and $\{ct_{\mathsf{dig}_i}\}_{i\in S}$ are implicitly encoded in an index-addressable form (*e.g.*, each entry as a pair (i, ct_{sd_i}) and (i, ct_{dig_i})) so that subsequent $\Pi_{\mathcal{R}_{\mathrm{val}}}.\mathsf{Vf}$ can unambiguously retrieve ct_{sd_i} and ct_{dig_i} for any $i \in S$.

Fig. 4: The aggregation interface for the server S.

Verification of server aggregation. Each verifier V_i for $i \in [n_{\mathrm{v}}]$ runs $\Pi_{\mathsf{SA}}.\mathsf{Vf}$ (Fig. 5) to validate the server's aggregation. It checks that the server selected aggregation set S has size at least $n_{\min}$; that for all $j \in S$, the client submission $(pk_{\mathrm{c},j}, \mathbf{y}_j, ct_{\mathsf{sd}_j}, ct_{\mathsf{dig}_j}, \pi_j, \sigma_{\mathrm{c},j})$ is valid; and that the aggregates are consistent,

i.e., $\sum_{j\in S}\mathbf{y}_j = \mathbf{y}$ and $\mathsf{AHE.EvalSum}(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \{ct_{\mathsf{sd}_j}\}_{j\in S}) = ct_{\mathsf{sd}}$.[4] On success, V_i certifies the aggregation by signing the aggregation message m with $sk_{\mathrm{v},i}$ via $\mathsf{SIG.Sign}$. Each honest verifier maintains local state and signs at most one aggregation message per protocol execution.

We denote the message certified by verifier V_i by m_i, *i.e.*, $m_i \coloneqq m$ for an honest verifier. This captures the fact that different verifiers may certify different messages under a malicious server or if the verifier itself is corrupted. Nevertheless, for a well-chosen certification threshold t_{v}, the following lemma ensures that at most one message m receives at least t_{v} valid signatures. Therefore, decryptors never face ambiguity about which aggregate to decrypt.

Lemma 2. *Let n_{v} be the verifier committee size and f_{v} the honest verifier fraction. Fix a threshold t_{v} satisfying:*

$$2t_{\mathrm{v}} - n_{\mathrm{v}} > (1 - f_{\mathrm{v}})n_{\mathrm{v}}. \tag{3.3}$$

Except with negligible probability, if there exists a message m that receives at least t_{v} valid signatures from distinct verifiers, there does not exist another message $m' \neq m$ that also receives at least t_{v} valid signatures from distinct verifiers.

Proof. Suppose for contradiction that there exist two distinct messages $m \neq m'$ *s.t.* each receives at least t_{v} valid signatures from distinct verifiers. Denote the respective index sets by:

$$I \coloneqq \{i \in [n_{\mathrm{v}}] : \mathsf{SIG.Vf}(pk_{\mathrm{v},i}, m, \sigma_{\mathrm{v},i}) = 1\},$$
$$I' \coloneqq \{i \in [n_{\mathrm{v}}] : \mathsf{SIG.Vf}(pk_{\mathrm{v},i}, m', \sigma'_{\mathrm{v},i}) = 1\}.$$

That is, $|I| \geq t_{\mathrm{v}}$ and $|I'| \geq t_{\mathrm{v}}$. By the intersection bound:

$$|I \cap I'| \geq |I| + |I'| - n_{\mathrm{v}} \geq 2t_{\mathrm{v}} - n_{\mathrm{v}}.$$

Then, by the threshold condition $2t_{\mathrm{v}} - n_{\mathrm{v}} > (1 - f_{\mathrm{v}})n_{\mathrm{v}}$:

$$|I \cap I'| > (1 - f_{\mathrm{v}})n_{\mathrm{v}}.$$

Since at most $(1-f_{\mathrm{v}})n_{\mathrm{v}}$ verifiers are corrupted, the intersection $I \cap I'$ contains at least one honest verifier. By the EUF-CMA security of SIG, except with negligible probability, both valid signatures under $pk_{\mathrm{v},i}$ were produced by this verifier. This contradicts the fact that an honest verifier signs at most one message per execution.

Partial decryption of masks. $\mathit{\Pi}_{\mathsf{SA}}.\mathsf{Dec}$ (Fig. 6) lets each decryptor D_i produce partial decryptions of the aggregated SHPRG seed ciphertext ct_{sd} and the hash seed ciphertexts $\{ct_{\mathsf{dig}_k}\}_{k\in S}$ contained in verifier certified aggregation messages. Concretely, from the set of received tuples $\{(pk_{\mathrm{v},j}, m_j, \sigma_{\mathrm{v},j})\}_{j\in[n_{\mathrm{v}}]}$, D_i selects any aggregation message m that admits at least t_{v} valid signatures. By Lemma 2, there exists at most one such message. The decryptor then computes $d_{\mathsf{sd},i} \leftarrow \mathsf{AHE.PartDec}(dk_{\mathsf{AHE},i}, ct_{\mathsf{sd}})$ and, for each $k \in S$, $d_{\mathsf{dig}_k,i} \leftarrow \mathsf{PKE.PartDec}(dk_{\mathsf{PKE},i}, ct_{\mathsf{dig}_k})$. Since the combination of these partial decryptions is publicly computable, they may be sent to the server or directly broadcast within $\mathit{\Pi}_{\mathsf{SA}}$.

[4] This verification step could be extended with a WillowFold-style proof-carrying data layer [36] that accumulates the submission validity and aggregation consistency checks, allowing each verifier to check one accumulated proof instead of reprocessing every selected submission.

$\Pi_{\mathsf{SA}}.\mathsf{Vf}(\mathsf{pp}, sk_{\mathrm{v},i}, (m, \{(pk_{\mathrm{c},j}, \mathbf{y}_j, ct_{\mathsf{sd}_j}, \pi_j, \sigma_{\mathrm{c},j})\}_{j\in S}))$

1. Parse $m = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_j}\}_{j\in S})$.
2. Denote $x_j \coloneqq (pk_{\mathrm{c},j}, \mathbf{y}_j, ct_{\mathsf{sd}_j}, ct_{\mathsf{dig}_j})$. Abort with $\perp$ if:
$$|S| < n_{\min} \vee \exists j \in S \ s.t.:$$
$$\Pi_{\mathcal{R}_{\mathrm{val}}}.\mathsf{Vf}(\mathsf{crs}, x_j, \pi_j) = 0 \vee \mathsf{SIG.Vf}(pk_{\mathrm{c},j}, (x_j, \pi_j), \sigma_{\mathrm{c},j}) = 0$$
$$\vee \sum_{j\in S} \mathbf{y}_j \neq \mathbf{y} \vee \mathsf{AHE.EvalSum}(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \{ct_{\mathsf{sd}_j}\}_{j\in S}) \neq ct_{\mathsf{sd}}.$$
3. Set $m_i \coloneqq m$ and sign with $\sigma_{\mathrm{v},i} \leftarrow \mathsf{SIG.Sign}(sk_{\mathrm{v},i}, m_i)$. Broadcast $(pk_{\mathrm{v},i}, m_i, \sigma_{\mathrm{v},i})$ to the decryptor committee $\{\mathsf{D}_j\}_{j\in[n_{\mathrm{d}}]}$.

Fig. 5: The verification interface for verifier V_i, $\forall i \in [n_{\mathrm{v}}]$.

$\Pi_{\mathsf{SA}}.\mathsf{Dec}(\mathsf{pp}, dk_{\mathsf{PKE},i}, dk_{\mathsf{AHE},i}, \{(pk_{\mathrm{v},j}, m_j, \sigma_{\mathrm{v},j})\}_{j\in[n_{\mathrm{v}}]})$[a]

1. For each distinct non-empty $m = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_k}\}_{k\in S})$, define the index set of verifiers with valid signatures:
$$I_m \coloneqq \{j \in [n_{\mathrm{v}}] : m_j = m \wedge \mathsf{SIG.Vf}(pk_{\mathrm{v},j}, m_j, \sigma_{\mathrm{v},j}) = 1\},$$
and abort with $\perp$ if $|I_m| < t_{\mathrm{v}}$ for all distinct messages.[b]
2. For $m = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_k}\}_{k\in S})$ *s.t.* $|I_m| \geq t_{\mathrm{v}}$, compute partial decryptions:
$$d_{\mathsf{sd},i} \leftarrow \mathsf{AHE.PartDec}(dk_{\mathsf{AHE},i}, ct_{\mathsf{sd}}),$$
$$d_{\mathsf{dig}_k,i} \leftarrow \mathsf{PKE.PartDec}(dk_{\mathsf{PKE},i}, ct_{\mathsf{dig}_k}), \forall k \in S.$$
3. Broadcast $(m_i \coloneqq m, d_{\mathsf{sd},i}, \{d_{\mathsf{dig}_k,i}\}_{k\in S})$ within Π_{SA}.[c]

[a] $\{(pk_{\mathrm{v},j}, m_j, \sigma_{\mathrm{v},j})\}_{j\in[n_{\mathrm{v}}]}$ may contain empty entries for verifiers that do not certify any aggregation message.
[b] By Eq. 3.3, equivalently, $t_{\mathrm{v}} \geq \lfloor(1 - f_{\mathrm{v}}/2)n_{\mathrm{v}}\rfloor + 1$.
[c] We similarly write m_i to denote the message instance processed by D_i.

Fig. 6: The decryption interface for decryptor D_i, $\forall i \in [n_{\mathrm{d}}]$.

Final delivery. As a publicly executable algorithm, $\Pi_{\mathsf{SA}}.\mathsf{Deliver}$ (Fig. 7) combines the decryptors' partial decryptions and recovers the final aggregation result. Similar to the partial decryption stage, it first identifies an aggregation message m that receives at least t_{d} decryptor responses. By Lemma 2, all honest decryptors process the same verifier certified message m. Therefore, $\Pi_{\mathsf{SA}}.\mathsf{Deliver}$ combines partial decryptions by this unique message. We choose the decryptor threshold as:
$$(1 - f_{\mathrm{d}})n_{\mathrm{d}} < t_{\mathrm{d}} \leq f_{\mathrm{d}} n_{\mathrm{d}}, \tag{3.4}$$
where the lower bound prevents corrupted decryptors from decrypting any ciphertext on their own, while the upper bound ensures that honest decryptors suffice for reconstruction. The remaining step is straightforward: $\Pi_{\mathsf{SA}}.\mathsf{Deliver}$ combines the corresponding partial decryptions to recover the aggregate SH-PRG seed sd' and the hash seeds $\{\mathsf{dig}'_k\}_{k\in S}$, and removes the resulting masks from $\mathbf{y}$ to obtain the input aggregate $\mathbf{x}'$.

Security analysis. We prove that Π_{SA} securely realizes $\mathcal{F}_{\mathrm{Agg}}$ with non-selective abort against a static malicious adversary $\mathcal{A}$. The proof proceeds by constructing a simulator $\mathcal{S}$ and showing, through a sequence of hybrids, that $\mathcal{A}$'s view in the

$\Pi_{\mathsf{SA}}.\mathsf{Deliver}(\mathsf{pp}, ek_{\mathsf{PKE}}, ek_{\mathsf{AHE}}, \{(m_i, d_{\mathsf{sd},i}, \{d_{\mathsf{dig}_k,i}\}_{k\in S_i})\}_{i\in[n_\mathrm{d}]})$[a]

1. For each distinct non-empty m, define the decryptor index set $I_m \coloneqq \{i \in [n_\mathrm{d}] : m_i = m\}$, and abort with $\perp$ if $|I_m| < t_\mathrm{d}$ for all distinct messages.[b]
2. For m *s.t.* $|I_m| \geq t_\mathrm{d}$, parse $m = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_k}\}_{k\in S})$ and define D_{sd} as:
$$\{d_{\mathsf{sd},i} : i \in I_m \wedge \mathsf{AHE.PartVf}(ek_{\mathsf{AHE}}, ct_{\mathsf{sd}}, d_{\mathsf{sd},i}) = 1\},$$
and, for each $k \in S$, define D_{dig_k} as:
$$\{d_{\mathsf{dig}_k,i} : i \in I_m \wedge \mathsf{PKE.PartVf}(ek_{\mathsf{PKE}}, ct_{\mathsf{dig}_k}, d_{\mathsf{dig}_k,i}) = 1\}.$$
3. Abort with $\perp$ if $|D_{\mathsf{sd}}| < t_\mathrm{d} \vee \bigvee_{k\in S}(|D_{\mathsf{dig}_k}| < t_\mathrm{d})$.
4. Recover the aggregate SHPRG seed and hash seeds:
$$\mathsf{sd}' \leftarrow \mathsf{AHE.Combine}(ek_{\mathsf{AHE}}, ct_{\mathsf{sd}}, D_{\mathsf{sd}}),$$
$$\mathsf{dig}'_k \leftarrow \mathsf{PKE.Combine}(ek_{\mathsf{PKE}}, ct_{\mathsf{dig}_k}, D_{\mathsf{dig}_k}), \forall k \in S.$$
5. Output the aggregation result:
$$\mathbf{x}' \coloneqq \mathbf{y} - \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}') - \sum_{k\in S} \mathsf{H}(\mathsf{dig}'_k).$$

[a] S_i denotes the aggregation set in m_i; $\{(m_i, d_{\mathsf{sd},i}, \{d_{\mathsf{dig}_k,i}\}_{k\in S_i})\}_{i\in[n_\mathrm{d}]}$ may also contain empty entries.
[b] The threshold t_d follows Eq. 3.4.

Fig. 7: The final delivery interface for all participants of Π_{SA}.

real execution is computationally indistinguishable from its view in the ideal execution with $\mathcal{S}$. The full proof is deferred to Appendix C.

Theorem 1. *Let* H *be a hash function,* SIG *be a signature scheme that satisfies EUF-CMA (Def. 3),* PRG *be a leakage-resilient SHPRG (Def. 1),* PKE *be a TPKE that satisfies IND-CPA and robustness (Def. 6 and 8),* AHE *be a TAHE that satisfies IND-CPA, simulation security, and robustness (Def. 6, 7, and 8),* $\Pi_{\mathcal{R}_{\mathrm{val}}}$ *be an SE-NIZKAoK (Def. 9, 10, 11, and 12). Let the certification threshold* t_v *and the decryption threshold* t_d *satisfy Eq. 3.3 and 3.4, respectively.* Π_{SA} *securely realizes* $\mathcal{F}_{\mathrm{Agg}}$ *with non-selective abort in the ROM against a static malicious adversary corrupting any number of clients, the server, up to a* $(1 - f_\mathrm{v})$ *fraction of verifiers, and up to a* $(1-f_\mathrm{d})$ *fraction of decryptors, where* $f_\mathrm{v}, f_\mathrm{d} > 2/3$.

Remark 2 (Reduced relation and partial extraction). When input validation is left outside $\mathcal{F}_{\mathrm{Agg}}$ and $\mathbf{x}_i$ ranges over the entire additive input space $\mathbb{F}^\ell$, the masking equality in Eq. 3.2 imposes no additional constraint on the public instance. We may retain the same instance x_i and prove only the two encryption equalities, using the witness $(\mathsf{sd}_i, r_{\mathsf{sd}_i}, \mathsf{dig}_i, r_{\mathsf{dig}_i})$ with the same domain restrictions. The proof circuit thus omits SHPRG expansion and hashing.

For Thm. 1, full witness extraction can further be relaxed to partial simulation extractability [10, Appendix B.1]: the extractor need only recover $(\mathsf{sd}_i, \mathsf{dig}_i)$ for which admissible encryption randomness satisfying both equalities exists. The simulation access and freshness conditions remain unchanged. The simulator reconstructs the client input as:
$$\mathbf{x}_i = \mathbf{y}_i - \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i) - \mathsf{H}(\mathsf{dig}_i),$$

and does not require the encryption randomness. Any input constraints must still be enforced on $\mathbf{x}_i$.

4 AlgOssa: Decentralized Secure Aggregation

The Ossa protocol Π_{SA} provides a standalone, one-shot secure aggregation core in a single-server, committee-aided setting. Its security guarantees, however, do not ensure that an aggregation result is eventually delivered. The designated server is the sole aggregation proposer, and its failure or refusal to proceed can halt the execution. Moreover, the phase-based communication model provides no mechanism for coordinating progress under network delays, replacing an unavailable server, or selecting among proposals constructed from different views of the client submissions. The verifier and decryptor committees are also preassigned rather than selected by the protocol. These limitations motivate replacing the single server with a dynamically selected set of potential leaders and using consensus to finalize one of their aggregation proposals.

Overview. Accordingly, we extend Ossa to AlgOssa by executing its interfaces over an Algorand-style backbone [22,11]. Each consensus round of AlgOssa carries one aggregation instance of Ossa and proceeds through a sequence of consensus steps. Clients submit their inputs as in Ossa, while cryptographic sortition dynamically selects potential leaders and fresh step committees from the consensus node population. The client and consensus node populations are not required to coincide; when they are distinct, lightweight devices may participate solely as clients.

Each potential leader constructs a candidate aggregation using the server interface of Ossa. The step 2 committee, acting as the verifier committee of Ossa, validates the candidate proposals, and at most one eligible candidate can obtain a quorum of its votes. The remaining consensus steps finalize either a non-empty candidate or the canonical empty block.

The consensus chain is then leveraged to coordinate threshold decryption. At designated points, cryptographic sortition derives a new decryptor committee from the finalized chain state and the randomness of the preceding block. The selected committee establishes the threshold encryption keys before client submissions under those keys are accepted. Once consensus certifies a block (with a non-empty aggregation payload), the honest decryptors verify its certificate and corresponding full block before releasing their partial decryption shares. With sufficiently many valid shares, any node can reconstruct the aggregate using the delivery interface of Ossa. In this case, threshold decryption is authorized by non-empty finalization; we also discuss an earlier release point, under the weaker guarantee of reconstructing at most one aggregate per round, in Remark 3.

Liveness of the Algorand backbone ensures that each round terminates except with negligible probability. If there are sufficiently many valid client submissions (cf. the $n_{\min}$ requirement of Ossa), each round additionally finalizes and delivers a non-empty aggregation with probability bounded below by a positive constant. A round may nevertheless finalize the canonical empty block, and hence aggre-

gates are not delivered at a fixed per-round cadence. Across repeated rounds, this yields an *eventual delivery guarantee*.

4.1 AlgOssa Protocol Model

The corruption model, honest fraction, and PKI assumptions are inherited directly from Ossa (Sec. 3.1). We present AlgOssa in the bounded synchronous network model [22], extending its timing parameters to account for the computation required by Ossa.[5]

Bounded synchronous network. Consensus nodes communicate over authenticated point-to-point channels. Each honest node has a local clock, and all honest clocks progress at the same rate, although their clock values need not be synchronized. The entry conditions for each round and step are specified by the protocol, and every timer is measured relative to the corresponding local entry time.

A message propagated by an honest node is assumed delivered to all honest nodes within a bounded time determined by its size. Small messages are delivered within time δ, whereas large messages are delivered within time Λ, where $\delta \leq \Lambda$. The adversary may determine the order and exact delivery times of messages subject to these bounds. No delivery guarantee is imposed on messages sent by corrupted nodes.

Computation aware timing. The network bounds δ and Λ apply from the time at which message propagation begins. Unlike ordinary consensus votes, constructing and validating an aggregation proposal requires non-negligible computation.[6]

Let κ^{s_1} denote an upper bound on the time required by an honest potential leader to construct an aggregation proposal, including the execution of $\Pi_{\mathsf{SA}}.\mathsf{Aggregate}$. Let κ^{s_2} denote an upper bound on the time required by an honest node to validate a complete proposal, including the execution of $\Pi_{\mathsf{SA}}.\mathsf{Vf}$. The corresponding protocol timers extend the original Algorand deadlines by these computation bounds along the affected execution paths. All other local computation is treated as negligible. The computation bounds are deployment parameters and must be conservatively chosen to cover honest execution at the supported system size.

Ephemeral step keys. Additionally, we follow the unique signature and ephemeral key model of Algorand [22]. For each round r and step s, a node uses a step-specific signing key for its prescribed signatures and erases the secret key afterward. Hence, corrupting an honest node after erasure does not enable the adversary to produce a conflicting message for that round and step.

[5] Recent work analyzes the deployed Algorand protocol under the weaker network assumption of partial synchrony [11]. That protocol includes recovery mechanisms absent from the presentation of [22] adopted here. Our integration of Ossa may also be applied to the deployed protocol by incorporating the additional computation into its timing parameters. We leave a formal analysis of this instantiation for future work.

[6] Algorand [22, Sec. 5.2] treats computation as negligible relative to network delay.

4.2 The AlgOssa Protocol

This section specifies the AlgOssa protocol Π_{DSA}. For ease of reference, we first recall the notation inherited from Ossa and Algorand, and then introduce the notation for the decryption committee selected in each key epoch.

- $n_{\mathrm{c}}, f_{\mathrm{c}}, n_{\min}$: the number of potential clients, the honest fraction of clients, and the minimal aggregation set size, respectively, where $n_{\min} \geq (1 - f_{\mathrm{c}})n_{\mathrm{c}} + 2$ (cf. Eq. 3.1).
- $(sk_{\mathrm{c},i}, pk_{\mathrm{c},i}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$: the long-term key pair of client i; the client is identified by $pk_{\mathrm{c},i}$.
- $r \in \mathbb{N}$ and $s \in \mathbb{N}^*$: the round and step indices, respectively, where steps are indexed within each round.
- M: the maximum number of steps in a round.
- $\mathsf{H} : \mathcal{D} \to \mathcal{R}$: a random oracle; define, for any $x \in \mathcal{D}$:
$$\dot{\mathsf{H}}(x) \coloneqq \mathrm{int}(\mathsf{H}(x))/|\mathcal{R}| \in [0, 1).$$
- n: the number of consensus nodes, which may vary across rounds as membership changes.
- $f > 2/3$: the assumed honest fraction of consensus nodes.
- $(sk_i, pk_i) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$: the long-term key pair of node i; the node is identified by pk_i.
- $(sk_i^{r,s}, pk_i^{r,s}) \leftarrow \mathsf{SIG.KGen}(1^\lambda)$: an ephemeral key pair of node i in round r step s.
- $\mathrm{sig}_i(m)$ and $\mathrm{SIG}_i(m) \coloneqq (pk_i, m, \mathrm{sig}_i(m))$: a signature on message m under pk_i and its message-retrievable tuple.
- $\mathrm{esig}_i(m)$ and $\mathrm{ESIG}_i(m) \coloneqq (pk_i^{r,s}, m, \mathrm{esig}_i(m))$: a signature on message m under $pk_i^{r,s}$ and its message-retrievable tuple.
- ℓ^r: the leader of round r.
- B^r: the block of round r, defined as:
$$B^r \coloneqq \begin{cases} (r, \mathsf{H}(B^{r-1}), \mathrm{SIG}_{\ell^r}(Q^{r-1}), agg) & \text{if } B^r \text{ is non-empty,} \\ B^r_\epsilon \coloneqq (r, \mathsf{H}(B^{r-1}), Q^{r-1}, \emptyset) & \text{if } B^r \text{ is empty,} \end{cases}$$
where agg is the aggregation message specified in Fig. 4. The head $\mathrm{Head}(B^r)$ consists of the first three components.
- Q^r: the seed of round r, defined as:
$$Q^r \coloneqq \begin{cases} \mathsf{H}(r, \mathrm{SIG}_{\ell^r}(Q^{r-1})) & \text{if } B^r \text{ is non-empty,} \\ \mathsf{H}(r, Q^{r-1}) & \text{if } B^r \text{ is empty.} \end{cases}$$
- PK^r: the set of consensus node public keys determined by the initial PKI state and any membership updates finalized in blocks $B^0, \ldots, B^{r-1}$.
- $p_{\mathrm{l}}^r, p_{\mathrm{v}}^{r,s} \in (0, 1]$: the probabilities of selecting a node as a potential leader or a step-s committee member in round r, respectively. Both are chosen so that the corresponding sets are non-empty except with negligible probability.
- $k \in \mathbb{N}^*$: the look-back parameter, chosen sufficiently large that the adversary cannot predict Q^{r-1} when PK^{r-k} is fixed, except with negligible probability [22, Lemma 5.11].
- PK^{-1}, B^{-1}, Q^{-1}: the initial consensus node public key set, public bootstrap block, and initial round seed, respectively. Once the initial PKI state is fixed,

let PK^{-1} be its registered consensus node public keys, sample $Q^{-1} \leftarrow\!\!\$\ \mathcal{R}$, and define:

$$B^{-1} \coloneqq (-1, \perp, Q^{-1}, \emptyset).$$

The block B^{-1} and seed Q^{-1} initialize round 0. For $r < k$, the look-back set PK^{r-k} is interpreted as PK^{-1}.

- $cred_i^{r,s} \coloneqq \mathrm{SIG}_i(r, s, Q^{r-1})$: the round r step s consensus credential of node i.
- $PL^r \coloneqq \{i \in [n] : pk_i \in PK^{r-k} \wedge \dot{\mathsf{H}}(cred_i^{r,1}) \le p_{\mathrm{l}}^r\}$: the set of potential leaders of round r. The selected leader is:

$$\ell^r \coloneqq \arg\min_{i \in PL^r} \mathsf{H}(cred_i^{r,1})$$

with ties broken lexicographically by pk_i.

- $SV^{r,s} \coloneqq \{i \in [n] : pk_i \in PK^{r-k} \wedge \dot{\mathsf{H}}(cred_i^{r,s}) \le p_{\mathrm{v}}^{r,s}\}$: the set of step $s > 1$ verifiers of round r step s. Define $SV^{r,1} \coloneqq PL^r$.
- $t_{\mathrm{v}} : \mathbb{R} \times (0, 1] \to \mathbb{N}^*$: the vote threshold function. Denote:

$$t_{\mathrm{v}}^{r,s} \coloneqq t_{\mathrm{v}}(p_{\mathrm{v}}^{r,s}|PK^{r-k}|, f)$$

the configured vote threshold for round r step s. The concrete instantiation of this function, including the sampling slack, is derived in the consensus analysis below.

- $CERT^r$: a certificate for B^r. For a non-empty block finalized at step $u = 5 + 3q$, $q \in \mathbb{N}$, it contains the matching step 1 header and at least $t_{\mathrm{v}}^{r,u-1}$ valid step $(u-1)$ messages from distinct eligible verifiers, all endorsing $(0, v)$ for $v = (\mathsf{H}(B^r), \ell^r)$. Certificates for the same block may contain different sets of supporting messages. A valid $CERT^r$ deterministically finalizes B^r and determines Q^r. If B^r satisfies $agg^r \neq \emptyset$, then $CERT^r$ additionally authorizes post-consensus reconstruction.
- $e \in \mathbb{N}$: the decryption key epoch. It does not affect consensus round numbering. Let r_e denote the first round whose submissions use the epoch e encryption keys; $r_0 \coloneqq 0$.
- $p_{\mathrm{d}}^e \in (0, 1]$: the probability of selecting a node as a decryptor for epoch e.
- $cred_i^{e,r_e,0} \coloneqq \mathrm{SIG}_i(e, r_e, 0, Q^{r_e-1})$: the epoch e decryptor credential of node i.
- $DC^e \coloneqq \{i \in [n] : pk_i \in PK^{r_e-k} \wedge \dot{\mathsf{H}}(cred_i^{e,r_e,0}) \le p_{\mathrm{d}}^e\}$: the decryptor committee for epoch e.
- $n_{\mathrm{d}}^e \coloneqq |DC^e|$: the number of decryptors selected for epoch e.
- $t_{\mathrm{d}} : \mathbb{R} \times (0, 1] \to \mathbb{N}^*$: the decryption threshold function. Denote $t_{\mathrm{d}}^e \coloneqq t_{\mathrm{d}}(p_{\mathrm{d}}^e|PK^{r_e-k}|, f)$ its value for epoch e.

We organize the presentation of Π_{DSA} into three components: protocol setup and submission of masked inputs, the consensus loop, and post-consensus reconstruction.

Protocol setup and submission of masked inputs. The setup of Π_{DSA} consists of the one-time $\Pi_{\mathsf{DSA}}.\mathsf{Setup}$ and the per-epoch $\Pi_{\mathsf{DSA}}.\mathsf{EpSetup}$. Together with $\Pi_{\mathsf{DSA}}.\mathsf{Submit}$, these interfaces operate outside the consensus loop. The initial setup and submission largely inherit their counterparts in Π_{SA}, whereas $\Pi_{\mathsf{DSA}}.\mathsf{EpSetup}$ establishes the threshold decryption parameters for each key epoch. They are presented in Fig. 8, Fig. 9, and Fig. 10, respectively.

$\Pi_{\mathsf{DSA}}.\mathsf{Setup}$ is executed once to establish the public parameters and bootstrap state $(PK^{-1}, B^{-1}, Q^{-1})$.

A key epoch e is a reference interval for threshold decryption along the globally ordered chain. Before the first round r_e using the epoch e keys, $\Pi_{\mathsf{DSA}}.\mathsf{EpSetup}$ derives DC^e from the existing blockchain state, namely PK^{r_e-k} and the sortition seed Q^{r_e-1} derived from the preceding certified block. For $e = 0$, it instead uses the bootstrap state (PK^{-1}, Q^{-1}). The presentation models $\Pi_{\mathsf{DSA}}.\mathsf{EpSetup}$ as a centralized procedure that sets up PKE and AHE over DC^e with threshold t_{d}^e and publishes dpp^e before accepting submissions under these keys. Alternatively, it can be realized by a DKG assisted by the chain: the finalized chain fixes DC^e, orders the public DKG transcript, and certifies dpp^e, while each selected decryptor contributes fresh private randomness and derives its decryption shares. The procedure is repeated at each key epoch to refresh the encryption keys and decryption shares, while consensus round numbering continues globally.

$\Pi_{\mathsf{DSA}}.\mathsf{Submit}$ invokes $\Pi_{\mathsf{SA}}.\mathsf{Submit}$ under $(ek^e_{\mathsf{PKE}}, ek^e_{\mathsf{AHE}})$ with execution identifier (e, r). The signature authenticates the submission and binds it to its key epoch and target round. We impose no restrictions on a client's inputs across rounds and require no synchronized submission deadline. A valid submission is eligible for inclusion if a potential leader receives it before fixing its round r pool.

$\Pi_{\mathsf{DSA}}.\mathsf{Setup}(1^\lambda, n_{\mathrm{c}}, n, n_{\min}, f, M, k, p_{\mathrm{l}}, p_{\mathrm{v}}, t_{\mathrm{v}}, \mathcal{R}_{\mathrm{val}})$[a]

1. Execute $\mathsf{pp}_{\mathsf{PRG}} \leftarrow \mathsf{PRG.Setup}(1^\lambda)$ and $(\mathsf{crs}, \mathsf{td}) \leftarrow \Pi_{\mathcal{R}_{\mathrm{val}}}.\mathsf{Setup}(1^\lambda, \mathcal{R}_{\mathrm{val}})$.
2. For each client $j \in [n_{\mathrm{c}}]$ and node $i \in [n]$, obtain $pk_{\mathrm{c},j}$ and pk_i, where $\mathsf{SIG.KGen}$ is run locally.
3. Register $(\{pk_{\mathrm{c},j}\}_{j\in[n_{\mathrm{c}}]}, \{pk_i\}_{i\in[n]})$ in the PKI and fix the initial set PK^{-1}.
4. After PK^{-1} is fixed, initialize Q^{-1} and B^{-1}.
5. Set the public protocol parameters:
$$\mathsf{pp} \coloneqq (n_{\mathrm{c}}, n, n_{\min}, f, M, k, p_{\mathrm{l}}, p_{\mathrm{v}}, t_{\mathrm{v}}, \mathsf{pp}_{\mathsf{PRG}}, \mathsf{crs}, PK^{-1}, B^{-1}).$$
Send pp to all participants of Π_{DSA}.

[a] p_{l} and p_{v} denote the configured families of selection probabilities. The values specified for round r and step s are p_{l}^r and $p_{\mathrm{v}}^{r,s}$, respectively.

Fig. 8: The one-time setup procedure.

The consensus loop. Each round of Π_{DSA} retains the proposal, graded consensus (GC), and binary Byzantine agreement (BBA) structure of Algorand [22]. The integration of Π_{SA} is confined to steps 1 and 2: step 1 constructs an aggregation block, and step 2 validates its aggregation before casting the first GC vote. Step 3 follows the original second GC step, with its deadline extended to account for aggregation verification.

Helper verification predicates. We define two helper predicates, $\mathsf{CredVf}^{r,s}$ and $\mathsf{MsgVf}^{r,s}$ with $s \geq 1$, for verifying credentials and step messages, respectively.

$\mathsf{CredVf}^{r,s}$ verifies both the credential signature and the node's eligibility for the corresponding round and step.

$\Pi_{\mathsf{DSA}}.\mathsf{EpSetup}(e, r_e, p_{\mathrm{d}}^e, t_{\mathrm{d}}, Q^{r_e-1})$

1. If $e = 0$, use Q^{-1}.
2. Each node $i \in [n]$ computes and propagates $cred_i^{e,r_e,0}$. Collect the selected decryptor set DC^e.
3. Execute:
$$(\mathsf{pp}_{\mathsf{PKE}}^e, ek_{\mathsf{PKE}}^e, \{dk_{\mathsf{PKE},i}^e\}_{i\in DC^e}) \leftarrow \mathsf{PKE.Setup}(1^\lambda, n_{\mathrm{d}}^e, t_{\mathrm{d}}^e)$$
$$(\mathsf{pp}_{\mathsf{AHE}}^e, ek_{\mathsf{AHE}}^e, \{dk_{\mathsf{AHE},i}^e\}_{i\in DC^e}) \leftarrow \mathsf{AHE.Setup}(1^\lambda, n_{\mathrm{d}}^e, t_{\mathrm{d}}^e)$$
4. Register $(ek_{\mathsf{PKE}}^e, ek_{\mathsf{AHE}}^e)$ in the PKI. For each $i \in DC^e$, send $(dk_{\mathsf{PKE},i}^e, dk_{\mathsf{AHE},i}^e)$ privately to node i.
5. Set the public decryption parameters:
$$\mathsf{dpp}^e := (e, r_e, p_{\mathrm{d}}^e, t_{\mathrm{d}}, DC^e, \mathsf{pp}_{\mathsf{PKE}}^e, \mathsf{pp}_{\mathsf{AHE}}^e).$$
6. Send dpp^e to all participants before accepting client submissions encrypted under the epoch e keys.[a]

[a] We omit pp and the applicable dpp^e from all subsequent procedures.

Fig. 9: The centralized decryption setup for key epoch e.

$\Pi_{\mathsf{DSA}}.\mathsf{Submit}(e, r, sk_{\mathrm{c},j}, ek_{\mathsf{PKE}}^e, ek_{\mathsf{AHE}}^e, \mathbf{x}_j)$

1. Run $\Pi_{\mathsf{SA}}.\mathsf{Submit}(sk_{\mathrm{c},j}, ek_{\mathsf{PKE}}^e, ek_{\mathsf{AHE}}^e, \mathbf{x}_j)$ with execution identifier (e, r). Denote the signed output by $\mathrm{SIG}_{\mathrm{c},j}(e, r, x_j, \pi_j)$.
2. Propagate $sub_j := \mathrm{SIG}_{\mathrm{c},j}(e, r, x_j, \pi_j)$ to consensus nodes as client j's submission for round r.

Fig. 10: The submission interface for client $j \in [n_{\mathrm{c}}]$ in epoch e.

– $\mathsf{CredVf}^{r,s}(cred_i^{r,s})=1$ iff $\mathsf{SIG.Vf}(cred_i^{r,s})=1$ and:
$$pk_i \in PK^{r-k} \wedge \begin{cases} \dot{\mathsf{H}}(cred_i^{r,1}) \le p_{\mathrm{l}}^r, & s = 1, \\ \dot{\mathsf{H}}(cred_i^{r,s}) \le p_{\mathrm{v}}^{r,s}, & s > 1. \end{cases}$$

For a step 1 proposal, $\mathsf{MsgVf}^{r,1}$ further verifies the ephemeral signature and the aggregation transcript using $\Pi_{\mathsf{SA}}.\mathsf{Vf}$. Here, $\Pi_{\mathsf{SA}}.\mathsf{Vf}$ performs only its verification checks and omits its signing operation. We extend it to the canonical empty output by defining $\Pi_{\mathsf{SA}}.\mathsf{Vf}(\cdot, (\emptyset, \emptyset)) = 1$; a pair containing exactly one empty component is rejected. For subsequent steps, $\mathsf{MsgVf}^{r,s}$ verifies the signed consensus value, and additionally the BBA bit when $s \ge 4$.

– For a message $m_i^{r,s}$ parsed according to its step s format, $\mathsf{MsgVf}^{r,s}(m_i^{r,s}) = 1$ iff $\mathsf{CredVf}^{r,s}(cred_i^{r,s}) = 1$ and:
$$\begin{cases} \mathsf{SIG.Vf}(pk_i^{r,1}, \mathsf{H}(B_i^r, \mathsf{tr}_i), \mathrm{esig}_i(\mathsf{H}(B_i^r, \mathsf{tr}_i))) = 1 \\ \qquad \wedge \Pi_{\mathsf{SA}}.\mathsf{Vf}(\cdot, (agg_i, \mathsf{tr}_i)) = 1, & s = 1, \\ \mathsf{SIG.Vf}(\mathrm{ESIG}_i(v_i)) = 1, & s \in \{2, 3\}, \\ \mathsf{SIG.Vf}(\mathrm{ESIG}_i(b_i)) = 1 \wedge \mathsf{SIG.Vf}(\mathrm{ESIG}_i(v_i)) = 1, & s \ge 4. \end{cases}$$

Step 1: block and aggregation proposal. The $\Pi_{\mathsf{DSA}}.\mathsf{Step}_1$ interface is presented in Fig. 11. Upon entering round r, node i fixes a snapshot $Pool_i^r$ containing at most one authenticated submission from each client. The snapshots held by

different nodes need not coincide; hence, honest potential leaders may propose different valid aggregations. A selected potential leader invokes $\Pi_{\mathsf{SA}}.\mathsf{Aggregate}$ on its snapshot and places the resulting aggregation message agg_i in its candidate block. The accompanying transcript tr_i is attached only to the full proposal and is bound to the block by the leader's ephemeral signature. The leader propagates the proposal for validation and its smaller header separately for leader selection.

$\underline{\Pi_{\mathsf{DSA}}.\mathsf{Step}_1(Q^{r-1}, sk_i, sk_i^{r,1}, Pool_i)}$

1. Node i enters step 1 when it obtains $CERT^{r-1}$; denote this local entry time by T_i^r. Define the round r snapshot:
$$Pool_i^r \coloneqq \{sub_j : j \in [n_c] \wedge sub_j \in Pool_i \wedge \mathsf{SIG.Vf}(sub_j) = 1\},$$
containing at most one submission for each client j.
2. Compute $cred_i^{r,1}$. If $i \notin PL^r$, return $\perp$.
3. Otherwise, compute the aggregation output:
$$(agg_i, \mathsf{tr}_i) \leftarrow \Pi_{\mathsf{SA}}.\mathsf{Aggregate}(Pool_i^r),$$
where agg_i is the aggregation message and tr_i the accompanying transcript specified in Fig. 4. If $\Pi_{\mathsf{SA}}.\mathsf{Aggregate}$ returns $\perp$, set $(agg_i, \mathsf{tr}_i) \coloneqq (\emptyset, \emptyset)$.
4. Compute the candidate block:
$$B_i^r \coloneqq (r, \mathsf{H}(B^{r-1}), \mathrm{SIG}_i(Q^{r-1}), agg_i).$$
5. Compute the message:
$$m_i^{r,1} \coloneqq ((B_i^r, \mathsf{tr}_i), \mathrm{esig}_i(\mathsf{H}(B_i^r, \mathsf{tr}_i)), cred_i^{r,1}),$$
and destroy the ephemeral key $sk_i^{r,1}$.
6. Propagate $m_i^{r,1}$ and $(\mathrm{Head}(B_i^r), cred_i^{r,1})$.

Fig. 11: The proposal interface for node $i \in [n]$ in round r.

Steps 2 and 3: aggregation verification and GC. The interfaces are presented in Fig. 12 and Fig. 13, respectively.

In $\Pi_{\mathsf{DSA}}.\mathsf{Step}_2$, after collecting potential leader credentials, each node selects the credential with the smallest hash in its local view and waits for the corresponding full proposal.[7] If the proposal arrives before the deadline and satisfies $\mathsf{MsgVf}^{r,1}$, the node adopts $v_i = (\mathsf{H}(B_\ell^r), \ell)$; otherwise, it adopts $\perp$. The step 2 committee thus serves as the verifier committee of Π_{SA}, with its signed step 2 vote providing the verifier's endorsement inside consensus. The transcript is no longer required after this validation and is omitted from all subsequent messages.

Then, in $\Pi_{\mathsf{DSA}}.\mathsf{Step}_3$, a node adopts a value v upon receiving $t_{\mathrm{v}}^{r,2}$ valid step 2 messages supporting that value; otherwise, it adopts $\perp$ when the timer expires. Except with negligible probability, the committee selection and threshold parameters ensure that every such set of votes contains an honest step 2 verifier. Moreover, instantiating Lemma 2 for the fixed round r step 2 shows that at most one value can obtain this support. The node then forwards its adopted value if selected for the step 3 committee.

[7] Alternatively, nodes could select the smallest-hash credential among potential leaders with valid aggregation proposals, at the cost of potentially verifying multiple proposals.

$\underline{\Pi_{\mathsf{DSA}}.\mathsf{Step}_2(Q^{r-1}, sk_i, sk_i^{r,2})}$

1. Node i enters step 2 when it obtains $CERT^{r-1}$, at local time T_i^r, and waits at most:
$$\tau_2 \coloneqq \delta + \kappa^{s_1} + \Lambda.$$
During this wait, node i processes the two forms of step 1 messages.[a]
2. After waiting $2\delta + \kappa^{s_1}$, define the index set of collected valid potential leader credentials:
$$I_1^r \coloneqq \{j \in [n] : \mathsf{CredVf}^{r,1}(cred_j^{r,1}) = 1\},$$
where $cred_j^{r,1}$ may be obtained from either form of step 1 message.
3. If $I_1^r \neq \emptyset$, select the local leader:
$$\ell \coloneqq \arg\min_{j \in I_1^r} \mathsf{H}(cred_j^{r,1}).$$
4. If, before τ_2 ends, node i has received:
$$m_\ell^{r,1} \ s.t. \ \mathsf{MsgVf}^{r,1}(m_\ell^{r,1}) = 1,$$
stop waiting and set $v_i \coloneqq (\mathsf{H}(B_\ell^r), \ell)$. Otherwise, when τ_2 ends, set $v_i \coloneqq \perp$.
5. Compute $cred_i^{r,2}$. If $i \notin SV^{r,2}$, return $\perp$. Otherwise, set:
$$m_i^{r,2} \coloneqq (\mathrm{ESIG}_i(v_i), cred_i^{r,2}),$$
destroy $sk_i^{r,2}$, and propagate $m_i^{r,2}$.

[a]The deadline τ_2 limits proposal receipt; validation may finish afterward, within the κ^{s_2} allowance in step 3.

Fig. 12: The aggregation verification and first GC step for node $i \in [n]$ in round r.

$\underline{\Pi_{\mathsf{DSA}}.\mathsf{Step}_3(Q^{r-1}, sk_i, sk_i^{r,3})}$

1. Node i enters step 3 when it obtains $CERT^{r-1}$, at local time T_i^r, and waits at most:
$$\tau_3 \coloneqq \tau_2 + 2\delta + \kappa^{s_2}.$$
During this wait, node i processes step 2 messages.
2. For each distinct v, let:
$$I_v^{r,2} \coloneqq \{j \in [n] : v_j = v \in m_j^{r,2} \wedge \mathsf{MsgVf}^{r,2}(m_j^{r,2}) = 1\}.$$
3. If, before τ_3 ends, there exists v *s.t.*:
$$|I_v^{r,2}| \geq t_{\mathrm{v}}^{r,2},$$
stop waiting and set $v_i \coloneqq v$. Otherwise, when τ_3 ends, set $v_i \coloneqq \perp$.
4. Compute $cred_i^{r,3}$. If $i \notin SV^{r,3}$, return $\perp$. Otherwise, set:
$$m_i^{r,3} \coloneqq (\mathrm{ESIG}_i(v_i), cred_i^{r,3}),$$
destroy $sk_i^{r,3}$, and propagate $m_i^{r,3}$.

Fig. 13: The second GC interface for node $i \in [n]$ in round r.

The timers account explicitly for the two computations introduced by Π_{DSA}: τ_2 includes the aggregation cost κ^{s_1} in $\Pi_{\mathsf{DSA}}.\mathsf{Step}_1$ and the large message delay Λ, whereas τ_3 additionally includes the verification cost κ^{s_2} in $\Pi_{\mathsf{DSA}}.\mathsf{Step}_2$. Each later step exchanges small messages and adds at most 2δ to the cumulative deadline.

The remaining interfaces adapt those from Algorand [22]. Concretely, $\Pi_{\mathsf{DSA}}.\mathsf{Step}_4$ (Fig. 16) determines the GC grade and initializes BBA with bit 0 exactly when the grade is 2. For each $q \in \mathbb{N}$, $\Pi_{\mathsf{DSA}}.\mathsf{Step}_{5+3q}$ (Fig. 17), $\Pi_{\mathsf{DSA}}.\mathsf{Step}_{6+3q}$ (Fig. 18), and $\Pi_{\mathsf{DSA}}.\mathsf{Step}_{7+3q}$ (Fig. 19) execute the coin-fixed-to-0, coin-fixed-to-1, and coin-genuinely-flipped BBA steps, respectively. These steps repeat until an ending condition holds; see Appendix E for the full interfaces.

Consensus guarantee. We first give a concrete instantiation of the vote threshold function t_{v} based on the honest fraction f and establish the resulting quorum properties of the sampled step committees. For round r and step s, let:

$$\bar{n}^{r,s} \coloneqq p_{\mathrm{v}}^{r,s}|PK^{r-k}|$$

denote the expected committee size, and let $h^{r,s}$ and $c^{r,s}$ denote the numbers of honest and corrupted committee members, respectively. Each eligible node has a unique credential for (r, s, Q^{r-1}). Since H is modeled as a random oracle, the values $\dot{\mathsf{H}}(cred_i^{r,s})$ are independently uniform in $[0, 1)$ for these distinct credentials. Hence, each node whose public key belongs to PK^{r-k} is independently selected with probability $p_{\mathrm{v}}^{r,s}$.

Choose $0 < \varepsilon < (3f - 2)/6$. Since at least a f fraction of the eligible nodes are honest, standard concentration bounds imply:

$$h^{r,s} \geq (f - \varepsilon)\bar{n}^{r,s},\ c^{r,s} \leq (1 - f + \varepsilon)\bar{n}^{r,s},\ h^{r,s} + c^{r,s} \leq (1 + \varepsilon)\bar{n}^{r,s},$$

simultaneously for all $s \in [2\,..\,M]$, except with negligible probability, provided the expected committee sizes are sufficiently large. Define the vote threshold by:

$$t_{\mathrm{v}}^{r,s} \coloneqq \left\lfloor (1 - \frac{f}{2} + 2\varepsilon)\bar{n}^{r,s} \right\rfloor + 1.$$

It then follows, except with negligible probability, that:

$$h^{r,s} \geq t_{\mathrm{v}}^{r,s} \wedge h^{r,s} + 2c^{r,s} < 2t_{\mathrm{v}}^{r,s}. \tag{4.1}$$

Therefore, honest members alone can form a quorum, whereas corrupted members alone cannot. Moreover, since each honest member signs at most one value in a fixed round and step, two conflicting quorums cannot be formed.

Theorem 2 (Consensus security). *Fix a round $r \geq 0$, and suppose that all honest nodes enter it from the same finalized predecessor block B^{r-1} certified by $CERT^{r-1}$, taking B^{-1} as the predecessor when $r = 0$. Under the assumptions stated in Sec. 4.1, suppose that Eq. 4.1 holds for every $s \in [2\,..\,M]$ and that M is chosen so that the probability that BBA does not terminate by step M is negligible. The following properties hold except with negligible probability:*

1. Agreement. *All honest nodes ending round r certify either the canonical empty block or the same non-empty block hash.*
2. Aggregation validity. *If the finalized block B^r contains $agg^r \neq \emptyset$, then there exists a transcript tr s.t.:*
$$\Pi_{\mathsf{SA}}.\mathsf{Vf}(\cdot, (agg^r, \mathsf{tr})) = 1.$$
3. Liveness. *Regardless of the honesty of the selected leader ℓ^r, every honest node i certifies the round r outcome by its local time $T_i^r + \tau_M$.*

– *If ℓ^r is honest, then all honest nodes finalize the block B^r proposed by ℓ^r. If $Pool^r_{\ell^r}$ contains at least $n_{\min}$ distinct valid client submissions, then B^r contains $agg^r \neq \emptyset$.*
– *The selected leader ℓ^r is honest with probability at least:*

$$p_{\mathrm{h}} = f^2(1 + f - f^2).$$

Proof. The integration of Π_{SA} changes only the step 1 proposal and the validity check performed before a step 2 vote; the GC and BBA transitions remain unchanged. The deadlines $\{\tau_s\}_{s\in[2..M]}$ account for the additional aggregation and verification costs. Hence, under Eq. 4.1, the agreement and termination arguments of Algorand [22, Thm. 1] carry over.

If a finalized block contains $agg^r \neq \emptyset$, GC and BBA validity trace its value to a quorum of step 2 votes. By Eq. 4.1, this quorum contains an honest verifier, which votes only after accepting $\Pi_{\mathsf{SA}}.\mathsf{Vf}(\cdot, (agg^r, \mathsf{tr}))$ for some transcript tr. If ℓ^r is honest and its pool contains at least $n_{\min}$ distinct valid submissions, $\Pi_{\mathsf{SA}}.\mathsf{Aggregate}$ produces $agg^r \neq \emptyset$, and the honest-leader argument finalizes its proposed block. Finally, the probability bound for an honest leader is exactly [22, Lemma 5.5].

The following proposition is a direct consequence of liveness.

Proposition 1 (Non-empty finalization). *Under the conditions of Thm. 2, suppose that the pool snapshot of each honest potential leader contains at least $n_{\min}$ distinct valid client submissions. Then, round r finalizes a block B^r satisfying $agg^r \neq \emptyset$ with probability at least $p_{\mathrm{h}} - \mathsf{negl}(\lambda)$.*

Post-consensus reconstruction. Let $\mathsf{CERTVf}^{e,r}(CERT^r, B^r)$ denote the public verification of a certificate and its corresponding block. It outputs 1 iff $CERT^r$ is a valid *ending-condition*-0 certificate for a *non-empty* round r block B^r, *i.e.*, all credentials and signatures contained in $CERT^r$ are valid, its certified value and step 1 header match B^r, and B^r contains a well-formed aggregation payload under the epoch e decryption parameters.

By Thm. 2, all round r certificates of honest nodes certify the same outcome. Although they may contain different supporting messages, every valid pair $(CERT^r, B^r)$ identifies the same aggregation payload agg^r. Hence, partial decryption is unambiguous.

For each $i \in DC^e$, whose committee membership was established during epoch setup using $cred_i^{e,r_e,0}$, $\Pi_{\mathsf{DSA}}.\mathsf{Dec}$ (Fig. 14) invokes the partial decryption algorithms of AHE and PKE on the ciphertexts in agg^r. It signs the resulting shares together with $(e, r, \mathsf{H}(B^r))$, binding them directly to the finalized block.

Any node may invoke $\Pi_{\mathsf{DSA}}.\mathsf{Deliver}$ (Fig. 15) after collecting responses from epoch decryptors. Denote their index set by $I_d^{e,r} \subseteq DC^e$. For each $i \in I_d^{e,r}$, retain at most one response and parse:

$$dmsg_i^{e,r} = \mathrm{SIG}_i(e, r, h_i, d_{\mathsf{sd},i}, \{(k, d_{\mathsf{dig}_k,i})\}_{k\in S}).$$

Define the valid response set:

$$D^{e,r} := \left\{ (agg^r, d_{\mathsf{sd},i}, \{d_{\mathsf{dig}_k,i}\}_{k\in S}) : \begin{array}{c} i \in I_d^{e,r} \wedge h_i = \mathsf{H}(B^r) \wedge \\ \mathsf{SIG.Vf}(dmsg_i^{e,r}) = 1 \end{array} \right\}. \tag{4.2}$$

$\Pi_{\mathsf{DSA}}.\mathsf{Dec}(e, r, sk_i, cred_i^{e,r_e,0}, dk^e_{\mathsf{PKE},i}, dk^e_{\mathsf{AHE},i}, CERT^r, B^r)$

1. Return $\perp$ if $i \notin DC^e \vee \mathsf{CERTVf}^{e,r}(CERT^r, B^r) \neq 1$.
2. Parse $agg^r = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_k}\}_{k\in S})$ and compute the partial decryptions prescribed by $\Pi_{\mathsf{SA}}.\mathsf{Dec}$:
$$d_{\mathsf{sd},i} \leftarrow \mathsf{AHE.PartDec}(dk^e_{\mathsf{AHE},i}, ct_{\mathsf{sd}}),$$
$$d_{\mathsf{dig}_k,i} \leftarrow \mathsf{PKE.PartDec}(dk^e_{\mathsf{PKE},i}, ct_{\mathsf{dig}_k}),\ \forall k \in S.$$
3. Propagate:
$$dmsg_i^{e,r} := \mathrm{SIG}_i(e, r, \mathsf{H}(B^r), d_{\mathsf{sd},i}, \{d_{\mathsf{dig}_k,i}\}_{k\in S}).$$

Fig. 14: Post-consensus partial decryption for node $i \in DC^e$.

If $|D^{e,r}| \geq t^e_{\mathrm{d}}$, $\Pi_{\mathsf{DSA}}.\mathsf{Deliver}$ invokes $\Pi_{\mathsf{SA}}.\mathsf{Deliver}$ on $D^{e,r}$. The latter verifies and combines the partial decryptions to reconstruct the aggregate. The same sampling analysis as for the vote committees shows that, except with negligible probability, honest decryptors alone can meet t^e_{d}, whereas corrupted decryptors cannot.

$\Pi_{\mathsf{DSA}}.\mathsf{Deliver}(e, r, CERT^r, B^r, \{dmsg_i^{e,r}\}_{i\in DC^e})$

1. Return $\perp$ if $\mathsf{CERTVf}^{e,r}(CERT^r, B^r) \neq 1$.
2. Define $D^{e,r}$ as Eq. 4.2. Execute:
$$\mathbf{x}^r \leftarrow \Pi_{\mathsf{SA}}.\mathsf{Deliver}(ek^e_{\mathsf{PKE}}, ek^e_{\mathsf{AHE}}, D^{e,r}),$$
where $\Pi_{\mathsf{SA}}.\mathsf{Deliver}$ uses decryption threshold t^e_{d}.

Fig. 15: Post-consensus delivery interface.

Remark 3 (Early decryption and at most one aggregate). A threshold of step 2 votes establishes the validity of an aggregation without finalizing it. By Eq. 4.1, at most one non-empty value can obtain this threshold in a fixed round and step. Thus, the protocol could authorize partial decryption immediately after a step 2 quorum is formed. Since all released shares would be bound to the uniquely supported block hash, this variant reconstructs at most one aggregation in each round. The resulting guarantee, however, is weaker: BBA may subsequently finalize the canonical empty block, in which case the reconstructed aggregate need not be the finalized consensus output. Our construction therefore waits for an ending-condition-0 certificate, ensuring that an aggregate is reconstructed only when its non-empty block is finalized.

5 Implementation and Evaluation for Ossa

We implement both instantiation routes of Π_{SA} described in Appendix D. All roles execute locally with in-memory state; networking and persistent state are outside the implementation. Python maintains protocol state and invokes compiled cryptographic libraries. Scalability experiments varying client count and vector length appear in Appendix F.3.

Cryptographic backends. The classical implementation uses BICYCL [17] for HSM-CL [18] and OpenSSL/libsecp256k1[8] for elliptic-curve operations used in DDH masking, threshold ElGamal, and Schnorr authentication. The PQ implementation uses threshold BFV [38] and C/C++ proof backends.
Proof scope. Classical submissions use the joint CL-ElGamal Fiat-Shamir proof for the reduced relation in Remark 2, including ciphertext admissibility and execution identifier binding. Delivery verifies partial decryption proofs before retaining at most one valid response per decryptor and checking the reconstruction threshold. For the PQ experiments, we retain Unruh-transformed ZKB++ [62,19] as the generic baseline and evaluate LNP22/LaZer [43,44] for encryption openings and Longfellow [31] for partial decryption.
Platform and evaluated profiles. Experiments ran on an Apple MacBook Air with an M3 processor and 16 GB of memory, using Python 3.14.3 and OpenSSL 3.6.1. Our classical instantiation targets 128-bit security. The classical evaluation uses 1,000 clients with 1,000-coordinate binary inputs and 100 decryptors per scheme. For Ossa, certification requires three of four verifiers, with corruption bound one. Each Ossa run executes all clients and decryptors and the three certifying verifiers sequentially. The PQ experiments use small, insecure BFV rings to assess proof feasibility.
Classical evaluation. Tab. 1 reports local computation costs and payload sizes for our classical implementation. For context, we include measurements from our extension of Willow's public code [8] and our OPA implementation [37]. Coverage differences are summarized in the table notes and detailed in Appendix F.1. We report means over three full runs per scheme, averaging per-party costs within each run. Timings include the implemented proofs, authentication, and verification checks, excluding network delays and process startup. Client preparation is reported separately.
Post-quantum feasibility. Tab. 2 reports mean proving and verification times and proof sizes over three proofs per configuration. Timings exclude parameter generation and circuit preparation; sizes use each implementation's binary proof encoding. LaZer jointly proves two BFV ciphertext openings under independent keys for AHE and PKE. Both decryption proofs enforce the same predicate: commitment checks, deterministic flooding noise derivation, and the partial decryption equation. Detailed proof configurations and preparation costs appear in Appendix F.2.

6 Conclusion

We answer the question posed in the introduction by separating the logically single-server aggregation interface from the identity of the party executing it. *Ossa* provides a modular, one-shot secure aggregation core with non-selective abort, while *AlgOssa* uses an Algorand-style consensus backbone to rotate protocol roles, finalize a single outcome, and authorize threshold reconstruction.

[8] `https://www.openssl.org/` and `https://github.com/bitcoin-core/secp256k1`

Tab. 1: Mean computation times and payload sizes for 1,000 clients, 1,000 coordinates, and 100 decryptors (helpers in OPA).

Metric	Scope	Ossa	Willow[a]	OPA
Client preparation[b]	Per client	19.6 ms	69.9 ms	—
Client submission	Per client	103 ms[c]	113 ms	1.24 s
Server verification[d]	Per client	29.1 ms	58.5 ms	0.647 ms
Server aggregation	Per client	0.963 ms	0.0191 ms	0.118 ms
Verifier certification	Per verifier	28.2 s	58.5 s	—
Decryptor response	Per decryptor	1.04 s	135 ms	1.01 s
Public reconstruction[e]	Full batch	12.1 s	6.34 s	1.81 s
Submission payload[f]	Per client	33.9 KiB	545 KiB	423 KiB
Response payload[f]	Per decryptor	129 KiB	460 KiB	19.6 KiB

A dash denotes no separate measurement. Rows may overlap and should not be summed. Aggregation costs are batch totals divided by the number of clients.

[a] We add Willow's specified auxiliary PRG masking, seed encryption, auxiliary-key sharing and release, and authorization to its public code. Measurements use one verifier and 100 honest decryptors, without dropout recovery.

[b] Initializes each client's arithmetic context and precomputed tables.

[c] Includes 31.5 ms of submission proof generation.

[d] Ossa includes proof and signature verification, structural checks, and admission bookkeeping. Willow checks submission proofs and nonce uniqueness. Our OPA implementation checks signatures and submission structure but omits malicious client sharing and masking consistency proofs.

[e] Ossa verifies and combines 17 valid responses with corruption bound $c = 16$. It returns the group-valued aggregate; numerical decoding is excluded and averages 45.1 ms per batch. Willow checks its implemented partial decryption proofs and combines all 100 additive contributions. OPA collects $r + 2c = 96$ responses with $r = 64$ and $c = 16$ for error-correcting reconstruction. All measured responses are honest.

[f] Payloads use a common fixed-width binary encoding. Outer protocol identifiers, signatures, framing, network overhead, and fanout are excluded; encrypted internal metadata remains included. Timings retain each implementation's protocol encoding.

Tab. 2: Mean PQ proof component costs.

Component[a]	Proof system	Prove	Verify	Size
Two-key opening	LNP22/LaZer	5.59 s	3.47 s	24.3 KiB
Partial decryption	Unruh–ZKB++	1.26 s	0.71 s	19.57 MiB
Partial decryption	Longfellow	4.09 s	2.71 s	12.29 MiB

[a] The two-key opening uses BFV ring dimension $N = 64$; both partial decryption experiments use $N = 8$.

Under the stated assumptions, this enables eventual delivery across repeated rounds without relying on a fixed server or committee. Our generic construction accommodates both classical and PQ instantiations. We evaluate the complete classical path and demonstrate the feasibility of the PQ route.

References

1. Abdalla, M., Benhamouda, F., Gay, R.: From single-input to multi-client inner-product functional encryption. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part III. LNCS, vol. 11923, pp. 552–582. Springer, Cham (Dec 2019). https://doi.org/10.1007/978-3-030-34618-8_19
2. Abdalla, M., Benhamouda, F., Kohlweiss, M., Waldner, H.: Decentralizing inner-product functional encryption. In: Lin, D., Sako, K. (eds.) PKC 2019, Part II. LNCS, vol. 11443, pp. 128–157. Springer, Cham (Apr 2019). https://doi.org/10.1007/978-3-030-17259-6_5
3. Addanki, S., Garbe, K., Jaffe, E., Ostrovsky, R., Polychroniadou, A.: Prio+: Privacy preserving aggregate statistics via boolean shares. In: Galdi, C., Jarecki, S. (eds.) SCN 22. LNCS, vol. 13409, pp. 516–539. Springer, Cham (Sep 2022). https://doi.org/10.1007/978-3-031-14791-3_23
4. Attema, T., Fehr, S., Klooß, M.: Fiat-Shamir transformation of multi-round interactive proofs (extended version). Journal of Cryptology **36**(4), 36 (Oct 2023). https://doi.org/10.1007/s00145-023-09478-y
5. Beaugrand, A., Castagnos, G., Laguillaumie, F.: Efficient succinct zero-knowledge arguments in the CL framework. Journal of Cryptology **38**(1), 13 (Jan 2025). https://doi.org/10.1007/s00145-024-09534-1
6. Bell, J., Gascón, A., Lepoint, T., Li, B., Meiklejohn, S., Raykova, M., Yun, C.: ACORN: Input validation for secure aggregation. In: Calandrino, J.A., Troncoso, C. (eds.) USENIX Security 2023. pp. 4805–4822. USENIX Association (Aug 2023), https://www.usenix.org/conference/usenixsecurity23/presentation/bell
7. Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure single-server aggregation with (poly)logarithmic overhead. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1253–1269. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417885
8. Bell-Clark, J., Gascón, A., Li, B., Raykova, M., Schoppmann, P.: Willow: Secure aggregation with one-shot clients. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VIII. LNCS, vol. 16007, pp. 285–318. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01913-4_9
9. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988). https://doi.org/10.1145/62212.62213
10. Benhamouda, F., Pointcheval, D.: Trapdoor smooth projective hash functions. Cryptology ePrint Archive, Report 2013/341 (2013), https://eprint.iacr.org/2013/341
11. Blum, E., Leung, D., Loss, J., Katz, J., Rabin, T.: Analyzing the real-world security of the algorand blockchain. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 830–844. ACM Press (Nov 2023). https://doi.org/10.1145/3576915.3623167
12. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1175–1191. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133982
13. Boneh, D., Boyen, X., Halevi, S.: Chosen ciphertext secure public key threshold encryption without random oracles. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS,

vol. 3860, pp. 226–243. Springer, Berlin, Heidelberg (Feb 2006). https://doi.org/10.1007/11605805_15

14. Boneh, D., Gennaro, R., Goldfeder, S., Jain, A., Kim, S., Rasmussen, P.M.R., Sahai, A.: Threshold cryptosystems from threshold fully homomorphic encryption. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part I. LNCS, vol. 10991, pp. 565–596. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96884-1_19
15. Boneh, D., Lewi, K., Montgomery, H.W., Raghunathan, A.: Key homomorphic PRFs and their applications. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 410–428. Springer, Berlin, Heidelberg (Aug 2013). https://doi.org/10.1007/978-3-642-40041-4_23
16. Boudgoust, K., Scholl, P.: Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part I. LNCS, vol. 14438, pp. 371–404. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8721-4_12
17. Bouvier, C., Castagnos, G., Imbert, L., Laguillaumie, F.: I want to ride my BICYCL : BICYCL implements CryptographY in CLass groups. Journal of Cryptology **36**(3), 17 (Jul 2023). https://doi.org/10.1007/s00145-023-09459-1
18. Braun, L., Castagnos, G., Damgård, I., Laguillaumie, F., Melissaris, K., Orlandi, C., Tucker, I.: An improved threshold homomorphic cryptosystem based on class groups. In: Galdi, C., Phan, D.H. (eds.) SCN 24, Part II. LNCS, vol. 14974, pp. 24–46. Springer, Cham (Sep 2024). https://doi.org/10.1007/978-3-031-71073-5_2
19. Chase, M., Derler, D., Goldfeder, S., Orlandi, C., Ramacher, S., Rechberger, C., Slamanig, D., Zaverucha, G.: Post-quantum zero-knowledge and signatures from symmetric-key primitives. In: Thuraisingham, B.M., Evans, D., Malkin, T., Xu, D. (eds.) ACM CCS 2017. pp. 1825–1842. ACM Press (Oct / Nov 2017). https://doi.org/10.1145/3133956.3133997
20. Chase, M., Kohlweiss, M., Lysyanskaya, A., Meiklejohn, S.: Verifiable elections that scale for free. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 479–496. Springer, Berlin, Heidelberg (Feb / Mar 2013). https://doi.org/10.1007/978-3-642-36362-7_29
21. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: Brickell, E.F. (ed.) CRYPTO'92. LNCS, vol. 740, pp. 89–105. Springer, Berlin, Heidelberg (Aug 1993). https://doi.org/10.1007/3-540-48071-4_7
22. Chen, J., Micali, S.: Algorand: A secure and efficient distributed ledger. Theor. Comput. Sci. **777**, 155–183 (2019). https://doi.org/10.1016/J.TCS.2019.02.001
23. Chen, R., Dong, Y., Liu, Y., Fan, T., Li, D., Guan, Z., Liu, J., Zhou, J.: Flock: Robust and privacy-preserving federated learning based on practical blockchain state channels. In: Long, G., Blumestein, M., Chang, Y., Lewin-Eytan, L., Huang, Z.H., Yom-Tov, E. (eds.) Proceedings of the ACM on Web Conference 2025, WWW 2025, Sydney, NSW, Australia, 28 April 2025- 2 May 2025. pp. 884–895. ACM (2025). https://doi.org/10.1145/3696410.3714666, https://doi.org/10.1145/3696410.3714666
24. Chotard, J., Dufour Sans, E., Gay, R., Phan, D.H., Pointcheval, D.: Decentralized multi-client functional encryption for inner product. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 703–732. Springer, Cham (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_24

25. Chowdhury, A.R., Guo, C., Jha, S., van der Maaten, L.: EIFFeL: Ensuring integrity for federated learning. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 2535–2549. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560611
26. Corrigan-Gibbs, H., Boneh, D.: Prio: Private, robust, and scalable computation of aggregate statistics. In: Akella, A., Howell, J. (eds.) 14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017. pp. 259–282. USENIX Association (2017), https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/corrigan-gibbs
27. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Fumy, W. (ed.) EUROCRYPT'97. LNCS, vol. 1233, pp. 103–118. Springer, Berlin, Heidelberg (May 1997). https://doi.org/10.1007/3-540-69053-0_9
28. Davis, H., Patton, C., Rosulek, M., Schoppmann, P.: Verifiable distributed aggregation functions. PoPETs **2023**(4), 578–592 (Oct 2023). https://doi.org/10.56553/popets-2023-0126
29. Devevey, J., Libert, B., Nguyen, K., Peters, T., Yung, M.: Non-interactive CCA2-secure threshold cryptosystems: Achieving adaptive security in the standard model without pairings. In: Garay, J. (ed.) PKC 2021, Part I. LNCS, vol. 12710, pp. 659–690. Springer, Cham (May 2021). https://doi.org/10.1007/978-3-030-75245-3_24
30. Feige, U., Shamir, A.: Zero knowledge proofs of knowledge in two rounds. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 526–544. Springer, New York (Aug 1990). https://doi.org/10.1007/0-387-34805-0_46
31. Frigo, M., shelat, a.: Anonymous credentials from ECDSA. Cryptology ePrint Archive, Report 2024/2010 (2024), https://eprint.iacr.org/2024/2010
32. Gao, G., Yu, H., Sun, Y., Hua, Z.: Sealagg: A lightweight auditing framework for one-shot secure aggregation against malicious servers. IACR Cryptol. ePrint Arch. **2026**, 869 (2026), https://eprint.iacr.org/2026/869
33. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology **20**(1), 51–83 (Jan 2007). https://doi.org/10.1007/s00145-006-0347-3
34. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on Computing **17**(2), 281–308 (Apr 1988)
35. Groth, J., Maller, M.: Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 581–612. Springer, Cham (Aug 2017). https://doi.org/10.1007/978-3-319-63715-0_20
36. Hafezi, H., Abbaszadeh, K., Gascón, A., Schoppmann, P., Raykova, M., Bünz, B.: Willowfold: Secure aggregation with a lightweight committee. IACR Cryptol. ePrint Arch. **2026**, 264 (2026), https://eprint.iacr.org/2026/264
37. Karthikeyan, H., Polychroniadou, A.: OPA: One-shot private aggregation with single client interaction and its applications to federated learning. In: Kalai, Y.T., Kamara, S.F. (eds.) CRYPTO 2025, Part VIII. LNCS, vol. 16007, pp. 319–353. Springer, Cham (Aug 2025). https://doi.org/10.1007/978-3-032-01913-4_10
38. Kim, W., Lee, C., Lee, J., Passelègue, A., Stehlé, D.: Asynchronous shamir-based threshold FHE: Smaller moduli and strong simulation security. Cryptology ePrint Archive, Paper 2026/973 (2026), https://eprint.iacr.org/2026/973

39. Li, H., Lin, H., Polychroniadou, A., Tessaro, S.: LERNA: Secure single-server aggregation via key-homomorphic masking. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part I. LNCS, vol. 14438, pp. 302–334. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8721-4_10
40. Liu, Z., Chen, S., Ye, J., Fan, J., Li, H., Li, X.: SASH: efficient secure aggregation based on SHPRG for federated learning. In: Cussens, J., Zhang, K. (eds.) Uncertainty in Artificial Intelligence, Proceedings of the Thirty-Eighth Conference on Uncertainty in Artificial Intelligence, UAI 2022, 1-5 August 2022, Eindhoven, The Netherlands. Proceedings of Machine Learning Research, vol. 180, pp. 1243–1252. PMLR (2022), https://proceedings.mlr.press/v180/liu22c.html
41. Lu, Z., Lu, S.: LZKSA: Lattice-based special zero-knowledge proofs for secure aggregation's input verification. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 573–587. ACM Press (Oct 2025). https://doi.org/10.1145/3719027.3744812
42. Lycklama, H., Burkhalter, L., Viand, A., Küchler, N., Hithnawi, A.: RoFL: Robustness of secure federated learning. In: 2023 IEEE Symposium on Security and Privacy. pp. 453–476. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179400
43. Lyubashevsky, V., Nguyen, N.K., Plançon, M.: Lattice-based zero-knowledge proofs and applications: Shorter, simpler, and more general. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part II. LNCS, vol. 13508, pp. 71–101. Springer, Cham (Aug 2022). https://doi.org/10.1007/978-3-031-15979-4_3
44. Lyubashevsky, V., Seiler, G., Steuer, P.: The LaZer library: Lattice-based zero knowledge and succinct proofs for quantum-safe privacy. In: Luo, B., Liao, X., Xu, J., Kirda, E., Lie, D. (eds.) ACM CCS 2024. pp. 3125–3137. ACM Press (Oct 2024). https://doi.org/10.1145/3658644.3690330
45. Ma, Y., Guo, Y., Karthikeyan, H., Polychroniadou, A.: Armadillo: Robust single-server secure aggregation for federated learning with input validation. In: Huang, C.Y., Chen, J.C., Shieh, S.P., Lie, D., Cortier, V. (eds.) ACM CCS 2025. pp. 2219–2233. ACM Press (Oct 2025). https://doi.org/10.1145/3719027.3765216
46. Ma, Y., Woods, J., Angel, S., Polychroniadou, A., Rabin, T.: Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In: 2023 IEEE Symposium on Security and Privacy. pp. 477–496. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179434
47. Melara, M.S., Blankstein, A., Bonneau, J., Felten, E.W., Freedman, M.J.: CONIKS: Bringing key transparency to end users. In: Jung, J., Holz, T. (eds.) USENIX Security 2015. pp. 383–398. USENIX Association (Aug 2015), https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/melara
48. Miao, Y., Liu, Z., Li, H., Choo, K.R., Deng, R.H.: Privacy-preserving byzantine-robust federated learning via blockchain systems. IEEE Trans. Inf. Forensics Secur. **17**, 2848–2861 (2022), https://doi.org/10.1109/TIFS.2022.3196274
49. Micciancio, D., Suhl, A.: Simulation-secure threshold PKE from LWE with polynomial modulus. CiC **1**(4), 2 (2024). https://doi.org/10.62056/a0zogy4e-
50. National Institute of Standards and Technology: Module-lattice-based digital signature standard. Federal Information Processing Standards Publication (FIPS) 204, U.S. Department of Commerce, Gaithersburg, MD (Aug 2024). https://doi.org/10.6028/NIST.FIPS.204, https://csrc.nist.gov/pubs/fips/204/final

51. Nguyen, D.D., Phan, D.H., Pointcheval, D.: Verifiable decentralized multi-client functional encryption for inner product. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Part V. LNCS, vol. 14442, pp. 33–65. Springer, Singapore (Dec 2023). https://doi.org/10.1007/978-981-99-8733-7_2
52. Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Malek, M., Huba, D.: Federated learning with buffered asynchronous aggregation. In: Camps-Valls, G., Ruiz, F.J.R., Valera, I. (eds.) Proceedings of The 25th International Conference on Artificial Intelligence and Statistics. Proceedings of Machine Learning Research, vol. 151, pp. 3581–3607. PMLR (28–30 Mar 2022), https://proceedings.mlr.press/v151/nguyen22b.html
53. Pass, R., Rosen, A.: New and improved constructions of non-malleable cryptographic protocols. In: Gabow, H.N., Fagin, R. (eds.) 37th ACM STOC. pp. 533–542. ACM Press (May 2005). https://doi.org/10.1145/1060590.1060670
54. Passelègue, A., Stehlé, D.: Low communication threshold fully homomorphic encryption. In: Chung, K.M., Sasaki, Y. (eds.) ASIACRYPT 2024, Part I. LNCS, vol. 15484, pp. 297–329. Springer, Singapore (Dec 2024). https://doi.org/10.1007/978-981-96-0875-1_10
55. Phong, L.T., Aono, Y., Hayashi, T., Wang, L., Moriai, S.: Privacy-preserving deep learning via additively homomorphic encryption. Cryptology ePrint Archive, Report 2017/715 (2017), https://eprint.iacr.org/2017/715
56. Qin, Z., Yan, X., Zhou, M., Deng, S.: Blockdfl: A blockchain-based fully decentralized peer-to-peer federated learning framework. In: Chua, T., Ngo, C., Kumar, R., Lauw, H.W., Lee, R.K. (eds.) Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024. pp. 2914–2925. ACM (2024). https://doi.org/10.1145/3589334.3645425, https://doi.org/10.1145/3589334.3645425
57. Rathee, M., Shen, C., Wagh, S., Popa, R.A.: ELSA: Secure aggregation for federated learning with malicious actors. In: 2023 IEEE Symposium on Security and Privacy. pp. 1961–1979. IEEE Computer Society Press (May 2023). https://doi.org/10.1109/SP46215.2023.10179468
58. Rathee, M., Zhang, Y., Corrigan-Gibbs, H., Popa, R.A.: Private analytics via streaming, sketching, and silently verifiable proofs. In: 2024 IEEE Symposium on Security and Privacy. pp. 3072–3090. IEEE Computer Society Press (May 2024). https://doi.org/10.1109/SP54263.2024.00245
59. Rothblum, G.N., Omri, E., Chen, J., Talwar, K.: PINE: Efficient verification of a euclidean norm bound of a secret-shared vector. In: Balzarotti, D., Xu, W. (eds.) USENIX Security 2024. USENIX Association (Aug 2024), https://www.usenix.org/conference/usenixsecurity24/presentation/rothblum
60. Schnorr, C.P.: Efficient signature generation by smart cards. Journal of Cryptology **4**(3), 161–174 (Jan 1991). https://doi.org/10.1007/BF00196725
61. Su, J., Bangalore, L., Berger, H., Yi, J., Castor, S., Sherr, M., Venkitasubramaniam, M.: SCIF: Privacy-preserving statistics collection with input validation and full security. PoPETs **2025**(3), 415–435 (Jul 2025). https://doi.org/10.56553/popets-2025-0106
62. Unruh, D.: Non-interactive zero-knowledge proofs in the quantum random oracle model. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 755–784. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46803-6_25
63. Wong, H.W.H., Ma, J.P.K., Chow, S.S.M.: Secure multiparty computation of threshold signatures made more efficient. In: NDSS 2024. The Internet Society (Feb / Mar 2024)

64. Zamani, M., Movahedi, M., Raykova, M.: RapidChain: Scaling blockchain via full sharding. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 931–948. ACM Press (Oct 2018). https://doi.org/10.1145/3243734.3243853

A Secure Aggregation Protocols

Tab. 3 compares representative secure aggregation protocols across their system model, interaction, threat model, dropout resilience, and output-delivery guarantees.

Tab. 3: Comparison of secure aggregation models and our protocols.

Protocol	**Server / committee**	**Interaction**	**Threat model**	**Dropout / delivery**
SecAgg [12]	1 S	4 rounds, 5 with consistency checking	C, S: semi-honest; active-privacy variant	Bounded; Not guaranteed
SecAgg+ [7]	1 S	Multi-round communication	C, S: semi-honest or semi-malicious	Bounded; Not guaranteed
RoFL [42]	1 S	Multi-round communication	S: semi-honest; C: malicious	Bounded; Not guaranteed
ACORN [6]	1 S	Multi-round communication	S: malicious for detection, semi-honest for robustness; C: malicious	Bounded; Client-robust variant
Armadillo [45]	1 S + client decryptor	3-round communication (independent of client count)	S: semi-honest; C, D: malicious	Bounded; Disruption-resistant
Prio [26]	$\geq$ 2 S (anytrust)	One-shot clients, servers interact	S: $\geq$ 1 honest for privacy, all honest for correctness; C: malicious	Client omissions; Not guaranteed
ELSA [57]	2 S (anytrust)	One-shot clients, servers interact	S: one malicious; C: malicious	Bounded; Not guaranteed
Flamingo [46]	1 S + rotating D committee	One-shot clients, reusable setup	C, S: static malicious; D: bounded corruption	Bounded; Detect-and-abort extension
LERNA [39]	1 S + fixed stateful committee	Non-committee clients: one-shot; 2 or 3 rounds overall	C, S: static malicious	Bounded; NSA
Willow [8]	1 S + stateful V, D committees	One-shot clients; interactive decryption	C, S: static malicious; V, D: bounded malicious	Bounded; NSA, GOD extension (under honest/semi-honest S)
OPA [37]	1 S + stateless committee	One-shot clients and helpers	C, S: static malicious; Committee: bounded malicious	Bounded; NSA
Ossa (this work)	1 S + preassigned V, D committees	One-shot clients; one-shot verifier and decryptor	C, S: static malicious; V, D: honest supermajority	Bounded; NSA
AlgOssa (this work)	Consensus-selected S, V, D	One-shot clients; multi-step consensus	C and node: static malicious, consensus honesty: $f > 2/3$	Bounded; Eventual delivery

SASH [40] uses SHPRG-based masking to reduce high-dimensional aggregation to masking-key aggregation over short seeds; this component may be instantiated with SecAgg or SecAgg+ [12,7].

"Bounded" denotes dropout tolerance subject to each protocol's stated participation thresholds.

"NSA" denotes non-selective abort, and "GOD" denotes guaranteed output delivery. LERNA and OPA are classified as NSA based on their ideal functionalities, although LERNA does not use this terminology and OPA describes its guarantee as "security with abort". "Not guaranteed" means that the cited work establishes no corresponding output-delivery guarantee; it does not, by itself, imply NSA. Flamingo provides detect-and-abort only through its robustness extension.

SecAgg, SecAgg+, SASH, and Flamingo do not provide application-level input validation in the compared constructions. The remaining protocols either provide some form of input validation or discuss it as a compatible or orthogonal extension, including through ACORN. Ossa and AlgOssa support a configurable validity relation through an SE-NIZKAoK, although we do not elaborate specific application-level predicates.

B Auxiliary Preliminaries

This section supplements Sec. 2 with additional details, including formal definitions of digital signatures and threshold public key encryption (TPKE).

B.1 Digital Signatures

A digital signature scheme consists of a tuple of algorithms $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vf})$.
- $\mathsf{KGen}(1^\lambda)$ takes as input the security parameter λ and outputs a secret-public key pair (sk, pk).
- $\mathsf{Sign}(sk, m)$ takes as input a secret key sk and a message m. It outputs a signature σ on m under sk.
- $\mathsf{Vf}(pk, m, \sigma)$ takes as input a public key pk, a message m, and a signature σ. It outputs 1 if σ is valid, or 0 otherwise.

We recall correctness and existential unforgeability under adaptive chosen message attacks (EUF-CMA) [34].

Definition 2 (Correctness). *A signature scheme is correct if the following holds for any λ and message m.*

$$\Pr\left[\mathsf{Vf}(pk, m, \sigma) = 1 \,\middle|\, \begin{array}{c}(sk, pk) \leftarrow \mathsf{KGen}(1^\lambda) \\ \sigma \leftarrow \mathsf{Sign}(sk, m)\end{array}\right] = 1$$

Definition 3 (EUF-CMA). *A signature scheme is EUF-CMA secure if, for any PPT adversary $\mathcal{A}$ with access to a signing oracle $\mathcal{O}_{\mathsf{Sign}}^{sk, Q_{\mathsf{Sign}}}$, the following probability is $\mathsf{negl}(\lambda)$ for any λ and $(sk, pk) \leftarrow \mathsf{KGen}(1^\lambda)$.*

$$\Pr\left[\begin{array}{c}\mathsf{Vf}(pk, m^*, \sigma^*) = 1 \\ \wedge m^* \notin Q_{\mathsf{Sign}}\end{array}\,\middle|\, \begin{array}{c}Q_{\mathsf{Sign}} := \emptyset \\ (m^*, \sigma^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sign}}}(pk)\end{array}\right],$$

where, on input m, the oracle $\mathcal{O}_{\mathsf{Sign}}$ returns $\sigma \leftarrow \mathsf{Sign}(sk, m)$ and records m in a queried set Q_{Sign}.

B.2 Threshold Public Key (Additively Homomorphic) Encryption

The following adapts the definitions of required properties, namely decryption correctness, partial verification correctness, threshold indistinguishability under chosen plaintext attack (IND-CPA), simulation security, and robustness from [20,14,29,16,54,49,63] to our TPKE and TAHE syntax. If a definition applies to both schemes, we state it jointly.

Definition 4 (Decryption Correctness). *A TPKE or TAHE scheme is decryption correct if the following probability is $1-\mathsf{negl}(\lambda)$ for any λ, n, t, $(\mathsf{pp}, ek, \{dk_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$, any $\{m_j\}_{j\in[k]} \in \mathcal{M}^k$, any admissible encryption randomness $\{r_j\}_{j\in[k]}$, and any set $S \subseteq [n] \wedge |S| \geq t$.*

$$\Pr\left[\begin{array}{c}\mathsf{Combine}(ek, \hat{ct}, \\ \{\phi(d_i)\}_{i\in S}) = \sum_{j\in[k]} m_j\end{array}\,\middle|\, \begin{array}{c}ct_j := \mathsf{Enc}(ek, m_j; r_j), \forall j \in [k] \\ \hat{ct} \leftarrow \mathsf{EvalSum}(ek, \{ct_j\}_{j\in[k]}) \\ d_i \leftarrow \mathsf{PartDec}(dk_i, \hat{ct}), \forall i \in S\end{array}\right]$$

This definition applies to TPKE by taking $k = 1$ and $\hat{ct} := ct_1$.

Definition 5 (Partial Verification Correctness). *A TPKE or TAHE scheme is partial verification correct if the following probability is* $1 - \mathsf{negl}(\lambda)$ *for any* λ, n, t, $(\mathsf{pp}, ek, \{dk_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$, *any* $\{m_j\}_{j\in[k]} \in \mathcal{M}^k$, *any admissible encryption randomness* $\{r_j\}_{j\in[k]}$, *and any* $i \in [n]$.

$$\Pr\left[\mathsf{PartVf}(ek, \hat{ct}, d_i) = 1 \;\middle|\; \begin{array}{c} ct_j \coloneqq \mathsf{Enc}(ek, m_j; r_j), \forall j \in [k] \\ \hat{ct} \leftarrow \mathsf{EvalSum}(ek, \{ct_j\}_{j\in[k]}) \\ d_i \leftarrow \mathsf{PartDec}(dk_i, \hat{ct}) \end{array}\right]$$

This definition applies to TPKE by taking $k = 1$ *and* $\hat{ct} \coloneqq ct_1$.

The security definitions of threshold encryption schemes vary across works. Here, we assume a static corruption model and consider the notions of threshold IND-CPA [49] and simulation security [20,14,54,63]. The former is relatively weak: it ignores threshold decryption and only requires that the (non-threshold) encryption scheme remains IND-CPA secure when the adversary compromises a subset of decryption keys of its choice. The latter requires partial decryptions to be simulatable from the public ciphertexts, corrupted decryption keys, and aggregate plaintext.

Definition 6 (Threshold IND-CPA). *A TPKE or TAHE scheme is IND-CPA secure if, for any* PPT *adversary* $\mathcal{A}$, *the following probability is at most* $1/2 + \mathsf{negl}(\lambda)$ *for any* λ, n, t.

$$\Pr\left[b' = b \;\middle|\; \begin{array}{c} (\mathsf{pp}, ek, \{dk_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t) \\ (S_\mathcal{A}, \mathsf{st}_0) \leftarrow \mathcal{A}(\mathsf{pp}, ek) \text{ s.t. } S_\mathcal{A} \subseteq [n] \wedge |S_\mathcal{A}| < t \\ (m_0, m_1, \mathsf{st}_1) \leftarrow \mathcal{A}(\mathsf{st}_0, \{dk_i\}_{i\in S_\mathcal{A}}) \text{ s.t. } |m_0| = |m_1| \\ b \leftarrow\!\$\ \{0,1\}; ct^* \leftarrow \mathsf{Enc}(ek, m_b); b' \leftarrow \mathcal{A}(\mathsf{st}_1, ct^*) \end{array}\right]$$

As pointed out in [54], the simulation security definition of [14, Definition 5.5] can be satisfied vacuously: the setup simulator may retain all secret key shares and use them to run the real decryption algorithms. To exclude this possibility while supporting verifiable decryption, we adapt the share simulatability framework of [20] and the threshold decryption simulation approach of [63]. The encryption key is honestly generated and supplied to the setup simulator without any decryption keys. The simulator generates the verification data, corrupted decryption keys, and simulation state, which subsequently allow it to simulate partial decryptions and their verification proofs from the public ciphertexts and aggregate plaintext. We require the joint setup and decryption view to be computationally indistinguishable from a real execution.

Definition 7 (Simulation Security). *For this definition, write the public setup* (pp, ek) *as* $(\mathsf{pub}_{\mathrm{enc}}, \mathsf{pp}_{\mathrm{vf}})$, *where* $\mathsf{pub}_{\mathrm{enc}}$ *contains the information required for encryption and homomorphic evaluation, and* $\mathsf{pp}_{\mathrm{vf}}$ *contains the remaining public information used to verify partial decryptions.*

A TAHE scheme is simulation secure if there exists a stateful PPT *simulator* $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ *s.t., for any* PPT *adversary* $\mathcal{A}$ *and any fixed corrupted set* $S_\mathcal{A} \subseteq [n]$ *with* $|S_\mathcal{A}| < t$, *the adversary's views in the following experiments are computationally indistinguishable.*

Both experiments honestly sample $(\mathsf{pp}, ek, \{dk_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$. *They initialize the adversary as follows:*

- *In the real experiment, give* $(\mathsf{pub}_{\mathrm{enc}}, \mathsf{pp}_{\mathrm{vf}}, \{dk_i\}_{i\in S_{\mathcal{A}}})$ *to* $\mathcal{A}$.
- *In the simulated experiment, compute:*

$$(\mathsf{pp}'_{\mathrm{vf}}, \{dk'_i\}_{i\in S_{\mathcal{A}}}, \mathsf{st}) \leftarrow \mathsf{Sim}_1(1^\lambda, n, t, \mathsf{pub}_{\mathrm{enc}}, S_{\mathcal{A}}),$$

  and give $(\mathsf{pub}_{\mathrm{enc}}, \mathsf{pp}'_{\mathrm{vf}}, \{dk'_i\}_{i\in S_{\mathcal{A}}})$ *to* $\mathcal{A}$. *The simulator receives no decryption keys from the honest setup.*

The adversary then outputs $\{(ct_j, m_j)\}_{j\in[k]}$, *where each* $m_j \in \mathcal{M}$ *and:*

$$\forall j \in [k], \exists r_j : ct_j = \mathsf{Enc}(ek, m_j; r_j),$$

with r_j *ranging over the admissible encryption randomness. Set* $\hat{ct} \coloneqq \mathsf{EvalSum}(ek, \{ct_j\}_{j\in[k]})$. *Retaining its state,* $\mathcal{A}$ *adaptively makes polynomially many decryption queries. For each query specifying a set* $S \subseteq [n]$, $\mathcal{A}$ *receives:*

- *In the real experiment,* $\{d_i\}_{i\in S}$, *where* $d_i \leftarrow \mathsf{PartDec}(dk_i, \hat{ct})$.
- *In the simulated experiment,* $\{d'_i\}_{i\in S}$, *where:*

$$(\{d'_i\}_{i\in S}, \mathsf{st}) \leftarrow \mathsf{Sim}_2(\mathsf{st}, \{ct_j\}_{j\in[k]}, \sum_{j\in[k]} m_j, S).$$

The updated st *remains private to the simulator.*

The simulated responses include all verification material. In the ROM, the simulator may program the random oracle used by the decryption proofs.

Finally, we adapt the robustness notion from [14, Definition 7.6] and incorporate the ϕ-projection from [29]. Intuitively, robustness ensures that even if the adversary learns all decryption keys, it cannot produce a valid (*w.r.t.* PartVf) partial decryption forgery (ct^*, d_i^*) on behalf of party i *s.t.* the forged share d_i^* does not match the honest partial decryption of ct^* (as measured under ϕ). This property further implies consistency [13], which requires that no adversary can prevent an eligible set of honest parties, *i.e.*, $S \subseteq [n] \wedge |S| \geq t$, from obtaining the decryption of a valid ciphertext through $\mathsf{Combine}$.

Definition 8 (Robustness). *A TPKE or TAHE scheme is robust if, for any* PPT *adversary, the following is* $\mathsf{negl}(\lambda)$ *for any* λ, n, t, *and* $(\mathsf{pp}, ek, \{dk_i\}_{i\in[n]}) \leftarrow \mathsf{Setup}(1^\lambda, n, t)$.

$$\Pr\left[\begin{array}{c}\mathsf{PartVf}(ek, ct^*, d_i^*) = 1 \wedge \\ \phi(d_i^*) \neq \phi(\mathsf{PartDec}(dk_i, ct^*))\end{array}\middle|\begin{array}{c}(ct^*, d_i^*, i) \\ \leftarrow \mathcal{A}(\mathsf{pp}, ek, \{dk_i\}_{i\in[n]})\end{array}\right]$$

B.3 Non-Interactive Arguments of Knowledge

We recall the definitions of completeness, zero-knowledge, knowledge soundness [30], and simulation extractability [53].

Definition 9 (Completeness). *A non-interactive argument protocol for a relation* $\mathcal{R}$ *satisfies completeness if the following probability is* 1 *for any* λ, $x \in L_{\mathcal{R}}$, *and* w *s.t.* $(x, w) \in \mathcal{R}$.

$$\Pr\left[\mathsf{Vf}(\mathsf{crs}, x, \pi) = 1 \middle| \begin{array}{c}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\end{array}\right]$$

Definition 10 (Zero-Knowledge). *A non-interactive argument for $\mathcal{R}$ is zero-knowledge if there exists a PPT simulator* Sim *s.t., for any* PPT *adversary $\mathcal{A}$, the following distribution ensembles are computationally indistinguishable for any λ and $(x, w) \in \mathcal{R}$.*

$$\mathcal{D}_0 \coloneqq \{(\mathsf{crs}, x, \pi_0) : \pi_0 \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\} \text{ and}$$
$$\mathcal{D}_1 \coloneqq \{(\mathsf{crs}, x, \pi_1) : \pi_1 \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, x)\}$$

$(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$ *is sampled in each distribution.*

Definition 11 (Knowledge Soundness). *A non-interactive argument for $\mathcal{R}$ is an argument of knowledge if, for any* PPT *adversary $\mathcal{A}$, there exists a PPT extractor* $\mathsf{Ext}^{\mathcal{A}}$ *s.t. the following probability is* $\mathsf{negl}(\lambda)$ *for any λ.*

$$\Pr\left[\begin{array}{c}\mathsf{Vf}(\mathsf{crs}, x, \pi) = 1 \\ \wedge (x, w) \notin \mathcal{R}\end{array}\middle|\begin{array}{c}(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ (x, \pi) \leftarrow \mathcal{A}(\mathsf{crs}) \\ w \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{crs}, x, \pi)\end{array}\right]$$

Definition 12 (Simulation Extractability). *A non-interactive argument for $\mathcal{R}$, with a PPT simulator* Sim*, satisfies simulation extractability if, for any PPT adversary $\mathcal{A}$, there exists a PPT extractor* $\mathsf{Ext}^{\mathcal{A}}$ *s.t. the following probability is* $\mathsf{negl}(\lambda)$ *for any λ and* $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R})$.

The adversary has access to a proof-simulation oracle $\mathcal{O}^{\mathsf{td}}_{\mathsf{Sim}}$. On input x, the oracle computes $\pi \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, x)$, records (x, π) in Q_{Sim}, and returns π.

$$\Pr\left[\begin{array}{c}\mathsf{Vf}(\mathsf{crs}, x, \pi) = 1 \\ \wedge (x, w) \notin \mathcal{R} \wedge (x, \pi) \notin Q_{\mathsf{Sim}}\end{array}\middle|\begin{array}{c}Q_{\mathsf{Sim}} \coloneqq \emptyset \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Sim}}}(\mathsf{crs}) \\ w \leftarrow \mathsf{Ext}^{\mathcal{A}}(\mathsf{crs}, x, \pi, Q_{\mathsf{Sim}})\end{array}\right]$$

C Proof of Thm. 1

We recall the statement as follows.

Theorem. *Let* H *be a hash function,* SIG *be a signature scheme that satisfies EUF-CMA (Def. 3),* PRG *be a leakage-resilient SHPRG (Def. 1),* PKE *be a TPKE that satisfies IND-CPA and robustness (Def. 6 and 8),* AHE *be a TAHE that satisfies IND-CPA, simulation security, and robustness (Def. 6, 7, and 8), $\Pi_{\mathcal{R}_{\mathrm{val}}}$ be an SE-NIZKAoK (Def. 9, 10, 11, and 12). Let the certification threshold t_{v} and the decryption threshold t_{d} satisfy Eq. 3.3 and 3.4, respectively. Π_{SA} securely realizes $\mathcal{F}_{\mathrm{Agg}}$ with non-selective abort in the ROM, against a static malicious adversary corrupting any number of clients, the server, up to a $(1 - f_{\mathrm{v}})$ fraction of verifier committee members, and up to a $(1 - f_{\mathrm{d}})$ fraction of decryptor committee members, where $f_{\mathrm{v}}, f_{\mathrm{d}} > 2/3$.*

Proof. Let $\mathcal{A}$ be a real-world adversary against the protocol Π_{SA}. Per static corruption, $\mathcal{A}$ fixes before protocol execution the corrupted client set C_{c}, corrupted verifier set C_{v}, and corrupted decryptor set C_{d}. Let $H_{\mathrm{c}} \coloneqq [n_{\mathrm{c}}] \setminus C_{\mathrm{c}}$, $H_{\mathrm{v}} \coloneqq [n_{\mathrm{v}}] \setminus C_{\mathrm{v}}$, and $H_{\mathrm{d}} \coloneqq [n_{\mathrm{d}}] \setminus C_{\mathrm{d}}$ denote the corresponding honest sets.

If $|H_{\mathrm{c}}| \leq 1$, no aggregation set can satisfy Eq. 3.1; hence, both $\mathcal{F}_{\mathrm{Agg}}$ and Π_{SA} abort. We therefore focus on $|H_{\mathrm{c}}| \geq 2$ and construct an ideal-world simulator $\mathcal{S}$

in the ROM (with a programmable random oracle). Throughout the simulation, $\mathcal{S}$ internally runs $\mathcal{A}$, relays all adversarially generated protocol messages, and emulates the honest parties' protocol messages as described below.

Setup. $\mathcal{S}$ generates $\mathsf{pp}_{\mathsf{PRG}}$, $(\mathsf{pp}_{\mathsf{PKE}}, ek_{\mathsf{PKE}}, \{dk_{\mathsf{PKE},i}\}_{i\in[n_\mathrm{d}]})$, and the client and verifier key pairs honestly. For AHE, it samples $(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \{dk_{\mathsf{AHE},i}\}_{i\in[n_\mathrm{d}]}) \leftarrow \mathsf{AHE.Setup}(1^\lambda, n_\mathrm{d}, t_\mathrm{d})$, writes $(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}})$ as $(\mathsf{pub}_{\mathrm{enc}}, \mathsf{pp}_{\mathrm{vf}})$, and runs the setup simulator (Def. 7):

$$(\mathsf{pp}'_{\mathrm{vf}}, \{dk'_{\mathsf{AHE},i}\}_{i\in C_\mathrm{d}}, \mathsf{st}_{\mathsf{AHE}}) \leftarrow \mathsf{AHE.Sim}_1(1^\lambda, n_\mathrm{d}, t_\mathrm{d}, \mathsf{pub}_{\mathrm{enc}}, C_\mathrm{d}). \tag{C.1}$$

$\mathcal{S}$ uses $(\mathsf{pub}_{\mathrm{enc}}, \mathsf{pp}'_{\mathrm{vf}})$ as the public AHE parameters and gives $\{dk'_{\mathsf{AHE},i}\}_{i\in C_\mathrm{d}}$ to the corrupted decryptors. For $\Pi_{\mathcal{R}_{\mathrm{val}}}$, it generates the common reference string crs together with the simulation trapdoor td. The remaining public parameters and public keys are published exactly as in the real protocol, and corrupted parties are given their corresponding secret states.

Client submission. For each honest client, $\mathcal{S}$ simulates a submission using a dummy input $\mathbf{x}_i = \mathbf{0}$. It samples sd_i and dig_i, computes the masked input $\mathbf{y}_i = \mathbf{0} + \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i) + \mathsf{H}(\mathsf{dig}_i)$, and encrypts both seeds as in $\Pi_{\mathsf{SA}}.\mathsf{Submit}$. It then sets $x_i \coloneqq (pk_{\mathrm{c},i}, \mathbf{y}_i, ct_{\mathsf{sd}_i}, ct_{\mathsf{dig}_i})$ and generates a simulated proof $\pi_i \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, x_i)$. Finally, $\mathcal{S}$ signs with $\sigma_{\mathrm{c},i} \leftarrow \mathsf{SIG.Sign}(sk_{\mathrm{c},i}, (x_i, \pi_i))$, sends $(x_i, \pi_i, \sigma_{\mathrm{c},i})$ to $\mathcal{A}$, and records $(\mathsf{sd}_i, \mathsf{dig}_i)$.

Aggregation and verification. When the server produces an aggregation transcript, $\mathcal{S}$ emulates each honest verifier V_i with $i \in H_\mathrm{v}$ by running $\Pi_{\mathsf{SA}}.\mathsf{Vf}$ honestly. For each submission included in the aggregation set S, it checks the proof and client signature under the registered public key. By EUF-CMA security of SIG, except with negligible probability, every accepted submission attributed to an honest client contains the instance and proof signed by that client for this execution. For each accepted submission from a corrupted client, $\mathcal{S}$ uses the extractor of $\Pi_{\mathcal{R}_{\mathrm{val}}}$ to recover and record the witness $(\mathbf{x}_j, \mathsf{sd}_j, r_{\mathsf{sd}_j}, \mathsf{dig}_j, r_{\mathsf{dig}_j})$. If the transcript is accepted, $\mathcal{S}$ signs the corresponding aggregation message m using the verifier's secret key and sends $(pk_{\mathrm{v},i}, m_i, \sigma_{\mathrm{v},i})$ to $\mathcal{A}$.

Certified aggregation set. If no aggregation message receives at least t_v valid verifier signatures, $\mathcal{S}$ sends $(\mathrm{deliver}, \mathrm{abort})$ to $\mathcal{F}_{\mathrm{Agg}}$ and simulates the aborting execution. Otherwise, by Lemma 2, there is a unique certified aggregation message $m = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_i}\}_{i\in S})$. For each corrupted client in S, $\mathcal{S}$ submits the input extracted from its submission in the certified transcript to $\mathcal{F}_{\mathrm{Agg}}$. If S is corrupted, $\mathcal{S}$ sends the finalize request with set S to $\mathcal{F}_{\mathrm{Agg}}$; otherwise, the honest S finalizes S. After receiving (chosen, S), it sends $(\mathrm{deliver}, \mathrm{continue})$. $\mathcal{F}_{\mathrm{Agg}}$ then returns $\mathbf{x}^* = \sum_{i\in S} \mathbf{x}_i$. $\mathcal{S}$ computes the honest clients' contribution:

$$\mathbf{x}^*_H \coloneqq \sum_{i\in S\cap H_\mathrm{c}} \mathbf{x}_i = \mathbf{x}^* - \sum_{i\in S\cap C_\mathrm{c}} \mathbf{x}_i. \tag{C.2}$$

For each $i \in S \cap C_\mathrm{c}$, $\mathcal{S}$ knows $\mathbf{x}_i$ from the extracted witness.

Programming the random oracle. Let $H \coloneqq S \cap H_\mathrm{c}$ be the honest client set included in the certified aggregation. By Lemma 1, H contains at least two honest clients. Except with negligible probability, there exists $i^* \in H$ *s.t.* dig_{i^*} has not been queried by $\mathcal{A}$ before it is opened. $\mathcal{S}$ reprograms the random oracle

value on dig_{i^*} by setting:

$$\mathsf{H}^*(\mathsf{dig}_{i^*}) \coloneqq \mathsf{H}(\mathsf{dig}_{i^*}) - \mathbf{x}^*_H, \tag{C.3}$$

where $\mathsf{H}(\mathsf{dig}_{i^*})$ denotes the simulator assigned hash mask for C_{i^*}'s dummy submission.

Partial decryption. For the certified aggregate seed ciphertext ct_{sd}, $\mathcal{S}$ uses the AHE decryption simulator (Def. 7) to produce partial decryptions for honest decryptors:

$$(\{d_{\mathsf{sd},i}\}_{i\in H_\mathrm{d}}, \mathsf{st}_{\mathsf{AHE}}) \leftarrow \mathsf{AHE.Sim}_2(\mathsf{st}_{\mathsf{AHE}}, \{ct_{\mathsf{sd}_j}\}_{j\in S}, \sum_{j\in S} \mathsf{sd}_j, H_\mathrm{d}). \tag{C.4}$$

For each certified hash seed ciphertext ct_{dig_j} with $j \in S$, $\mathcal{S}$ computes partial decryptions for each honest decryptor using $\{dk_{\mathsf{PKE},i}\}_{i\in H_\mathrm{d}}$ via PKE.PartDec. It then broadcasts the decryptor messages exactly as prescribed by Π_{SA}.Dec.

Delivery. Having committed to continue before learning $\mathbf{x}^*$, $\mathcal{S}$ runs the publicly executable Π_{SA}.Deliver algorithm on the simulated honest partial decryptions for the unique certified message with aggregation set S, together with any valid shares provided by $\mathcal{A}$ through corrupted decryptors. By Eq. 3.4, the honest decryptors alone provide at least t_d valid shares. For PKE, partial verification correctness, decryption correctness, and robustness ensure that filtering via PKE.PartVf and combining via PKE.Combine reconstruct the exact plaintexts $\{\mathsf{dig}_i\}_{i\in S}$ for all admissible encryption randomness. For AHE, these same properties ensure that AHE.PartVf and AHE.Combine reconstruct $\sum_{j\in S} \mathsf{sd}_j$ under an honest AHE setup and real partial decryptions; because Π_{SA}.Deliver is a publicly computable PPT function of the public parameters, ciphertexts, and partial decryptions, joint simulation security (Def. 7) guarantees that AHE.PartVf and AHE.Combine likewise reconstruct $\sum_{j\in S} \mathsf{sd}_j$ except with negligible probability under the simulated parameters $\mathsf{pp}'_{\mathrm{vf}}$ and shares $\{d_{\mathsf{sd},i}\}_{i\in H_\mathrm{d}}$ (cf. Hybrid$_2$). Therefore, except with negligible probability, Π_{SA}.Deliver reconstructs the masks and outputs $\mathbf{x}^*$. No further abort decision is made by $\mathcal{S}$.

We now show that the above simulation is computationally indistinguishable from the real-world execution of Π_{SA}. By Lemma 2, there is at most one aggregation message that receives t_v valid verifier signatures; hence, at most one aggregation message $m = (S, \mathbf{y}, ct_{\mathsf{sd}}, \{ct_{\mathsf{dig}_i}\}_{i\in S})$ is certified. When a certified message exists, Lemma 1 ensures that the selected honest client set $H = S \cap H_\mathrm{c}$ contains at least two clients; when no message is certified, the execution aborts prior to partial decryption, in which case we take $S \coloneqq \emptyset$ and $H \coloneqq \emptyset$ in the hybrids below.

The privacy claim across both certifying and aborting executions proceeds through the following sequence of hybrids.

Hybrid$_0$**.** This is the real-world execution of Π_{SA}.

Hybrid$_1$**.** For each $i \in H_\mathrm{c}$, we replace the NIZKAoK proof π_i with a simulated proof $\pi_i \leftarrow \mathsf{Sim}(\mathsf{crs}, \mathsf{td}, x_i)$. The view of $\mathcal{A}$ is indistinguishable from that in Hybrid$_0$ by zero-knowledge of $\Pi_{\mathcal{R}_{\mathrm{val}}}$ (Def. 10).

Hybrid$_2$**.** We replace the real AHE setup and partial decryptions with the simulated AHE experiment. The simulator samples $\mathsf{pub}_{\mathrm{enc}}$ via AHE.Setup and runs

$\mathsf{AHE.Sim}_1$ to obtain the simulated verification parameters $\mathsf{pp}'_{\mathrm{vf}}$, corrupted decryption keys $\{dk'_{\mathsf{AHE},i}\}_{i\in C_{\mathrm{d}}}$, and simulator state $\mathsf{st}_{\mathsf{AHE}}$ (cf. Eq. C.1). All client seed ciphertexts in this hybrid are generated under $\mathsf{pub}_{\mathrm{enc}}$, and the server computes the corresponding aggregate ciphertext using $\mathsf{AHE.EvalSum}$. If an aggregation message becomes certified on set S, $\mathcal{S}$ invokes $\mathsf{AHE.Sim}_2$ on $\mathsf{st}_{\mathsf{AHE}}$, the ciphertexts $\{ct_{\mathsf{sd}_j}\}_{j\in S}$, the aggregate seed $\sum_{j\in S}\mathsf{sd}_j$, and the honest decryptor set H_{d} (cf. Eq. C.4).

To obtain the aggregate seed, $\mathcal{S}$ uses the seeds it sampled for the honest clients in H. For corrupted clients in S, the witnesses $\{(\mathbf{x}_j, \mathsf{sd}_j, r_{\mathsf{sd}_j}, \mathsf{dig}_j, r_{\mathsf{dig}_j})\}_{j\in S\cap C_{\mathrm{c}}}$ are obtained by applying the knowledge extractor $\Pi_{\mathcal{R}_{\mathrm{val}}}.\mathsf{Ext}$ to the corresponding accepted proofs $\{\pi_j\}_{j\in S\cap C_{\mathrm{c}}}$. By simulation extractability of $\Pi_{\mathcal{R}_{\mathrm{val}}}$ (Def. 12), extraction fails only with negligible probability, even though $\mathcal{A}$ has previously received simulated proofs for honest-client submissions, and successful extraction guarantees that each ct_{sd_j} is a well-formed encryption under admissible randomness r_{sd_j}. Conditioned on successful extraction, the view of $\mathcal{A}$, including the publicly computable output of $\Pi_{\mathsf{SA}}.\mathsf{Deliver}$, is computationally indistinguishable from that in Hybrid_1 by the simulation security of AHE (Def. 7).

Hybrid_3**.** For each $i \in H_{\mathrm{c}}$, we replace ct_{sd_i} with an AHE encryption of a dummy seed 0, while we continue to run $\mathsf{AHE.Sim}_2$ as post-processing, supplying the aggregate seed $\sum_{j\in S}\mathsf{sd}_j$ whenever an aggregation message is certified. Since $\mathsf{AHE.Sim}_1$ receives only $\mathsf{pub}_{\mathrm{enc}}$ and no decryption keys from the honest $\mathsf{AHE.Setup}$, the view of $\mathcal{A}$ is computationally indistinguishable from that in Hybrid_2 by Threshold IND-CPA of AHE (Def. 6).

Hybrid_4**.** For each $i \in H_{\mathrm{c}}$, instead of sampling $\mathsf{H}(\mathsf{dig}_i) \leftarrow\!\$\ \mathbb{F}^\ell$ prior to submission and setting $\mathbf{y}_i \coloneqq \mathbf{x}_i + \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i) + \mathsf{H}(\mathsf{dig}_i)$, we sample $\mathbf{y}_i \leftarrow\!\$\ \mathbb{F}^\ell$ uniformly and independently at random during client submission and define the random oracle value at dig_i lazily for queries occurring after certification as:

$$\mathsf{H}(\mathsf{dig}_i) \coloneqq \mathbf{y}_i - \mathbf{x}_i - \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i). \tag{C.5}$$

If $\mathcal{A}$ queries H on dig_i for any $i \in H_{\mathrm{c}}$ before an aggregation message is certified (or before the execution halts in the no-certificate branch), or if $\mathsf{dig}_i = \mathsf{dig}_{i'}$ for distinct $i, i' \in H_{\mathrm{c}}$, the simulation aborts. Conditioned on this bad event not occurring, the joint distribution of $(\mathbf{y}_i, \mathsf{H}(\mathsf{dig}_i))$ for all $i \in H_{\mathrm{c}}$ is identical to that in Hybrid_3. Moreover, prior to certification, no honest decryptor executes $\mathsf{PKE.PartDec}$; a reduction to Threshold IND-CPA of PKE (Def. 6) that halts at the certification decision, and hence requires no PKE decryption queries, shows that replacing $\{ct_{\mathsf{dig}_i}\}_{i\in H_{\mathrm{c}}}$ with encryptions of 0 changes the probability of a pre-certification query to $\{\mathsf{dig}_i\}_{i\in H_{\mathrm{c}}}$ by at most $\mathsf{negl}(\lambda)$. When $\{ct_{\mathsf{dig}_i}\}_{i\in H_{\mathrm{c}}}$ encrypt 0, $\{\mathsf{dig}_i\}_{i\in H_{\mathrm{c}}}$ are independent of $\mathcal{A}$'s pre-certification view and are queried or collide with negligible probability. Therefore, the view of $\mathcal{A}$ is computationally indistinguishable from that in Hybrid_3. Note that in Hybrid_4, the submitted vectors $\{\mathbf{y}_i\}_{i\in H_{\mathrm{c}}}$ are i.i.d. uniform in $\mathbb{F}^\ell$ and independent of $\{\mathbf{x}_i, \mathsf{sd}_i\}_{i\in H_{\mathrm{c}}}$; in particular, in the no-certificate aborting branch, $\mathcal{A}$'s view is now completely independent of $\{\mathbf{x}_i\}_{i\in H_{\mathrm{c}}}$, and in the certifying branch, the server's selected set S, and hence $H = S \cap H_{\mathrm{c}}$, is independent of $\{\mathsf{sd}_i\}_{i\in H_{\mathrm{c}}}$.

Hybrid$_5$. For each unselected honest client $i \in H_c \setminus S$, we replace its post-certification random oracle value in Eq. C.5 with:

$$\mathsf{H}(\mathsf{dig}_i) \coloneqq \mathbf{y}_i - \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i),$$

keeping the submitted vector $\mathbf{y}_i$ fixed and effectively setting $\mathbf{x}_i = \mathbf{0}$ for all $i \in H_c \setminus S$, even if $\mathcal{A}$ subsequently learns dig_i after threshold decryption. In Hybrid$_4$, for every $i \in H_c \setminus S$, ct_{sd_i} encrypts 0, $\mathbf{y}_i$ and S are independent of sd_i, and sd_i is excluded from the aggregate seed $\sum_{j \in S} \mathsf{sd}_j$ passed to $\mathsf{AHE.Sim}_2$. By the PRG security of PRG (Def. 1), $\{\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)\}_{i \in H_c \setminus S}$ is computationally indistinguishable from i.i.d. uniform $\{\mathbf{u}_i\}_{i \in H_c \setminus S}$ in $\mathbb{F}^\ell$; since $\mathbf{x}_i + \mathbf{u}_i$ and $\mathbf{u}_i$ are identically distributed in $\mathbb{F}^\ell$, replacing $\mathbf{x}_i + \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)$ with $\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)$ is computationally indistinguishable.

Hybrid$_6$. If an aggregation message is certified, let $H = S \cap H_c$, where $|H| \geq 2$ by Lemma 1, and fix a pivot index $i^* \in H$. Keeping the already-submitted vectors $\{\mathbf{y}_i\}_{i \in H_c}$ fixed as sampled in Hybrid$_4$, we sample an independent uniform mask $\mathbf{u}_i \leftarrow\!\$\, \mathbb{F}^\ell$ for each $i \in H \setminus \{i^*\}$, define:

$$\mathbf{u}_{i^*} \coloneqq \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \sum_{i \in H} \mathsf{sd}_i) - \sum_{i \in H \setminus \{i^*\}} \mathbf{u}_i,$$

and set the post-certification random oracle values for $i \in H$ to:

$$\mathsf{H}(\mathsf{dig}_i) \coloneqq \mathbf{y}_i - \mathbf{x}_i - \mathbf{u}_i, \forall i \in H. \tag{C.6}$$

In Hybrid$_5$, by seed-homomorphism of PRG:

$$\begin{aligned}\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_{i^*}) = \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \sum_{i \in H} \mathsf{sd}_i) \\ - \sum_{i \in H \setminus \{i^*\}} \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i).\end{aligned}$$

Because $\{\mathbf{y}_i\}_{i \in H_c}$ and the selection of H are independent of $\{\mathsf{sd}_i\}_{i \in H}$, and $\{ct_{\mathsf{sd}_i}\}_{i \in H}$ encrypt 0, the view of $\mathcal{A}$ depends on $\{\mathsf{sd}_i\}_{i \in H}$ only through $\{\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)\}_{i \in H \setminus \{i^*\}}$ and the aggregate seed $\sum_{i \in H} \mathsf{sd}_i$ passed to $\mathsf{AHE.Sim}_2$. By a hybrid argument across $i \in H \setminus \{i^*\}$ invoking the leakage resilience of PRG under leakage of the sum of seeds (Def. 1), replacing $\{\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)\}_{i \in H \setminus \{i^*\}}$ with i.i.d. uniform $\{\mathbf{u}_i\}_{i \in H \setminus \{i^*\}}$ is computationally indistinguishable from Hybrid$_5$.

Hybrid$_7$. For each $i \in H \setminus \{i^*\}$, define the shifted mask $\mathbf{u}'_i \coloneqq \mathbf{x}_i + \mathbf{u}_i$. Since $\{\mathbf{u}_i\}_{i \in H \setminus \{i^*\}}$ are i.i.d. uniform in the additive group $\mathbb{F}^\ell$ and independent of $(\{\mathbf{x}_i\}_{i \in H}, \{\mathbf{y}_i\}_{i \in H_c}, \sum_{i \in H} \mathsf{sd}_i)$, the shifted tuple $\{\mathbf{u}'_i\}_{i \in H \setminus \{i^*\}}$ has the exact same i.i.d. uniform distribution. Substituting $\mathbf{u}_i = \mathbf{u}'_i - \mathbf{x}_i$ into Eq. C.6 yields:

$$\begin{aligned}\mathsf{H}(\mathsf{dig}_i) &= \mathbf{y}_i - \mathbf{u}'_i, \forall i \in H \setminus \{i^*\}, \\ \mathsf{H}(\mathsf{dig}_{i^*}) &= \mathbf{y}_{i^*} - \mathbf{x}_{i^*} - \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \sum_{i \in H} \mathsf{sd}_i) + \sum_{i \in H \setminus \{i^*\}} (\mathbf{u}'_i - \mathbf{x}_i) \\ &= \mathbf{y}_{i^*} - \mathbf{x}^*_H - (\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \sum_{i \in H} \mathsf{sd}_i) - \sum_{i \in H \setminus \{i^*\}} \mathbf{u}'_i),\end{aligned}$$

where $\mathbf{x}^*_H = \sum_{i \in H} \mathbf{x}_i$ (cf. Eq. C.2). That is, redistributing the honest inputs in H to set $\mathbf{x}_i = \mathbf{0}$ for $i \in H \setminus \{i^*\}$ and $\mathbf{x}_{i^*} = \mathbf{x}^*_H$ induces the exact same joint distribution on $(\{\mathbf{y}_i\}_{i \in H_c}, \{\mathsf{H}(\mathsf{dig}_i)\}_{i \in H_c}, \sum_{i \in H} \mathsf{sd}_i)$ as Hybrid$_6$.

Hybrid_8. Applying the leakage-resilient SHPRG hybrid argument of Hybrid_6 (Def. 1) in reverse, we replace the i.i.d. uniform tuple $\{\mathbf{u}'_i\}_{i\in H\setminus\{i^*\}}$ in Hybrid_7 back with $\{\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)\}_{i\in H\setminus\{i^*\}}$, so that:

$$\begin{aligned}&\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \sum_{i\in H} \mathsf{sd}_i) - \sum_{i\in H\setminus\{i^*\}} \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)\\ =&\mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_{i^*}).\end{aligned}$$

The post-certification random oracle values are now $\mathsf{H}(\mathsf{dig}_i) = \mathbf{y}_i - \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i)$ for all $i \in H_c \setminus \{i^*\}$ and $\mathsf{H}^*(\mathsf{dig}_{i^*}) = \mathbf{y}_{i^*} - \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_{i^*}) - \mathbf{x}^*_H$. Reversing the lazy random oracle sampling of Hybrid_4, which changes $\mathcal{A}$'s view with at most negligible probability by the pre-certification PKE IND-CPA bound established in Hybrid_4, each honest client $i \in H_c$ samples $\mathsf{H}(\mathsf{dig}_i) \leftarrow\!\$\ \mathbb{F}^\ell$ at submission time and sets $\mathbf{y}_i \coloneqq \mathbf{0} + \mathsf{PRG.Expand}(\mathsf{pp}_{\mathsf{PRG}}, \mathsf{sd}_i) + \mathsf{H}(\mathsf{dig}_i)$, while $\mathcal{S}$ reprograms only $\mathsf{H}^*(\mathsf{dig}_{i^*}) \coloneqq \mathsf{H}(\mathsf{dig}_{i^*}) - \mathbf{x}^*_H$ upon certification (cf. Eq. C.3). Thus, the view of $\mathcal{A}$ in Hybrid_8 is computationally indistinguishable from that in Hybrid_7.

Hybrid_9. Finally, for each $i \in H_c$, we replace the dummy seed encryption ct_{sd_i} from Hybrid_3 with $\mathsf{AHE.Enc}(\mathsf{pp}_{\mathsf{AHE}}, ek_{\mathsf{AHE}}, \mathsf{sd}_i; r_{\mathsf{sd}_i})$ as generated by $\mathcal{S}$. By Threshold IND-CPA of AHE (Def. 6), the view of $\mathcal{A}$ is computationally indistinguishable from that in Hybrid_8. Moreover, Hybrid_9 is identical to the ideal-world execution with $\mathcal{S}$. This concludes the proof.

D Concrete Instantiations

This section describes classical and PQ instantiation routes for the cryptographic primitives used by Ossa (and hence AlgOssa), together with distributed generation of their threshold encryption keys. These instantiations serve as the basis for our implementation and evaluation in Sec. 5.

Classical instantiation. We instantiate ordinary authentication with Schnorr signatures [60].[9]

For the SHPRG, we employ the DDH-based construction of [15]. For a prime-order group $\mathbb{G}$ of order q, its vector output form has seed space $\mathcal{K} = \mathbb{Z}_q$ and output space $\mathbb{G}^\ell$. Its exact seed homomorphism and DDH security yield the required pseudorandomness under leakage of the seed sum. We take $\mathbb{F}^\ell = \mathbb{G}^\ell$, write its group operation additively, and model H as a random oracle into this space.

For the encryption core, we instantiate TAHE with HSM-CL [18] and TPKE with verifiable threshold ElGamal [27], using Chaum-Pedersen proofs [21] to verify ElGamal partial decryptions. The HSM-CL plaintext space is chosen as $\mathbb{Z}_q$ to match the SHPRG seed group, while the ElGamal plaintext space contains $\mathcal{D}_{\mathsf{H}}$. Their encryption security gives threshold IND-CPA (Def. 6) under the assumptions of the respective constructions, and both satisfy decryption correctness (Def. 4) and partial verification correctness (Def. 5) over all admissible

[9] Schnorr and the ML-DSA scheme [50] used in the PQ route below cover authentication in Ossa. Sortition in AlgOssa separately requires the unique signature mechanism of its consensus layer.

encryption randomness. Their decryption proofs bind each accepted partial decryption to the corresponding public verification key; taking ϕ to discard the proof component yields the share uniqueness required by Def. 8. For simulation security (Def. 7), we adapt the HSM-CL setup and decryption simulators of [18]. We include the aggregate, unscaled public commitment from its DKG in $\mathsf{pub}_{\mathrm{enc}}$; the working encryption key is derived from this commitment as in the construction, while the per-decryptor verification keys belong to $\mathsf{pp}_{\mathrm{vf}}$. This representation allows Sim_1 to sample the corrupted shares and interpolate the simulated verification data in the exponent from $\mathsf{pub}_{\mathrm{enc}}$ while preserving the externally supplied encryption parameters. Its decryption simulator Sim_2 then uses the aggregate ciphertext and aggregate plaintext, together with ROM-simulated decryption proofs.

For submission proofs, the simulation-extractable zkSNARK of Groth and Maller (GM17) [35] supplies the required SE-NIZKAoK with full witness extraction. When input validity is not required, it can be applied to the reduced relation in Remark 2, proving only the two encryption equalities. Under the partial extraction relaxation in that remark, we combine the CL ciphertext-correctness argument of [5] with an ElGamal proof and apply Fiat-Shamir in the ROM. The joint proof binds the full instance, including the client identity and execution identifier, to its challenges and enforces the relation's admissibility conditions. Under static corruption, this binding separates simulated honest client proofs from corrupted client extraction targets. Together with honest-verifier zero knowledge and partial special soundness in the CL rough-order framework, this supports simulation and expected polynomial-time extraction in our standalone protocol proof [4].

PQ route. We instantiate ordinary authentication with ML-DSA [50] and the SHPRG with the LWR-based almost seed-homomorphic construction of [15], paired with the encoding and decoding of [37] to handle bounded rounding error.

For the encryption core, we use the Shamir-based threshold BFV construction of [38]. Two independent key instances provide TAHE and TPKE, respectively; the latter uses only encryption and decryption. The TAHE plaintext ring encodes the SHPRG seed coordinates so that plaintext addition preserves seed addition, while the TPKE plaintext space accommodates $\mathcal{D}_{\mathsf{H}}$. Threshold BGV is an alternative through the extension in [38, Sec. 3.3]. For simulation security, we use the standard simulation secure variant of [38]. Its setup simulator generates corrupted shares independently of the secret key, and its decryption simulator uses the public ciphertext and its plaintext [38, Thm. 3.2]. The simulation argument applies to ciphertexts satisfying the prescribed decryption-noise bound. We choose the encryption parameters and the admissible randomness bounds enforced by $\Pi_{\mathcal{R}_{\mathrm{val}}}$ so that every accepted aggregate, up to the maximum aggregation size, satisfies this noise bound.

Publicly verifiable decryption requires an additional layer. Following the commitment-and-proof approach of [14, Sec. 7.1], setup places hiding and binding commitments to each decryptor's secret share in $\mathsf{pp}_{\mathrm{vf}}$, and each partial decryption carries a non-interactive proof of its correct computation. To meet our

robustness notion, we additionally derive the bounded flooding noise deterministically from the ciphertext using a private PRF seed committed alongside the decryption share during setup. We apply to the PRF output a deterministic sampler that maps uniform bits to the prescribed bounded flooding distribution up to negligible statistical distance. The decryption proof checks both the partial decryption computation and this noise derivation against the public commitment, while ϕ discards the proof component.

For submission and decryption proofs, we retain Unruh-transformed ZKB++ [62,19] as a generic baseline, with simulation extractability in the *quantum* ROM (QROM) established in [19, Appendix F].[10] When input validity is not required, submission proofs use the reduced relation in Remark 2, retaining full witness extraction. In the *classical* ROM, we also consider LNP22 [43], implemented using LaZer [44], for submission proofs, and Longfellow [31] for decryption proofs. We evaluate their feasibility in Sec. 5 and leave a formal security analysis of the alternative proof integrations and augmented threshold decryption to future work.

Distributed key generation. After the decryptor committee and reconstruction threshold are fixed, let n denote the number of decryptors, t the reconstruction threshold, and c the maximum number of statically corrupted decryptors. By Eq. 3.4, these parameters satisfy $c < t \leq n - c$. The DKG generates degree-$(t-1)$ shares; hence, any t valid shares suffice for reconstruction.

For the classical route, the native HSM-CL DKG [18] and the GJKR DKG [33] provide distributed setup within their stated parameter regimes. Applying their security results directly requires $c \leq t - 1 < n/2$ [18, Thm. 11], [33, Thm. 1], which is permitted by the honest super-majority assumption in the decryptor committee.

More generally, for any reconstruction threshold $c < t \leq n - c$ and for the augmented PQ threshold BFV construction, the prescribed setup algorithms can be evaluated through honest-majority secure computation against $c < n/3$ static corruptions [9], revealing the public parameters and delivering each private share only to its recipient. This provides a generic distributed setup route under the required communication assumptions; it serves as an existence construction rather than an optimized lattice DKG.

For AlgOssa, the resulting public outputs are fixed in the finalized state of the applicable key epoch; for Ossa, they are included in the public parameters. The corresponding decryption key shares are distributed to the DKG participants. No submission under the resulting keys is accepted until these public outputs are fixed.

E The Remaining Steps of AlgOssa

This section presents the remaining consensus interfaces of Π_{DSA}. It specifies $\Pi_{\mathsf{DSA}}.\mathsf{Step}_4$ (Fig. 16) and the three recurring BBA interfaces $\Pi_{\mathsf{DSA}}.\mathsf{Step}_{5+3q}$,

[10] The security analysis for Thm. 1 in the QROM remains open.

$\Pi_{\mathsf{DSA}}.\mathsf{Step}_{6+3q}$, and $\Pi_{\mathsf{DSA}}.\mathsf{Step}_{7+3q}$ for $q \in \mathbb{N}$ (Fig. 17, 18, and 19). These interfaces follow the Algorand consensus loop [22] and are included for completeness.

$\underline{\Pi_{\mathsf{DSA}}.\mathsf{Step}_4(Q^{r-1}, sk_i, sk_i^{r,4})}$

1. Node i enters step 4 upon finishing step 3 and waits at most 2δ. Hence, the maximum total time from T_i^r is:
$$\tau_4 \coloneqq \tau_3 + 2\delta.$$
During this wait, node i processes step 3 messages.
2. For each distinct v, let:
$$I_v^{r,3} \coloneqq \{j \in [n] : v_j = v \in m_j^{r,3} \wedge \mathsf{MsgVf}^{r,3}(m_j^{r,3}) = 1\}.$$
3. If, before τ_4 ends, there exists $v \neq \perp$ *s.t.*:
$$|I_v^{r,3}| \geq t_{\mathrm{v}}^{r,3},$$
stop waiting and set $(v_i^*, g_i) \coloneqq (v, 2)$.
4. Otherwise, if, before τ_4 ends:
$$|I_\perp^{r,3}| \geq t_{\mathrm{v}}^{r,3},$$
stop waiting and set $(v_i^*, g_i) \coloneqq (\perp, 0)$.
5. Otherwise, when τ_4 ends, if there exists $v \neq \perp$ *s.t.*:
$$|I_v^{r,3}| \geq \lceil t_{\mathrm{v}}^{r,3}/2 \rceil,$$
set $(v_i^*, g_i) \coloneqq (v, 1)$. Otherwise, set $(v_i^*, g_i) \coloneqq (\perp, 0)$.
6. Set the BBA input:
$$b_i \coloneqq \begin{cases} 0, & g_i = 2, \\ 1, & \text{otherwise.} \end{cases}$$
7. Compute $cred_i^{r,4}$. If $i \notin SV^{r,4}$, return v_i^* without propagating a step 4 message. Otherwise, set:
$$m_i^{r,4} \coloneqq (\mathrm{ESIG}_i(b_i), \mathrm{ESIG}_i(v_i^*), cred_i^{r,4}),$$
destroy $sk_i^{r,4}$, propagate $m_i^{r,4}$, and return v_i^*.

Fig. 16: The GC output and first BBA step for node $i \in [n]$ in round r.

F Additional Experimental Details

This section supplements Sec. 5 with setup and key-reuse considerations, PQ proof configurations and preparation costs, and scalability experiments. Detailed measurement procedures and reproduction instructions are provided in our code repository.

F.1 Setup and Key Reuse

Ossa. The implementation executes the native HSM-CL DKG [18] and GJKR DKG [33] for threshold ElGamal locally and sequentially. A separate benchmark of three fault-free setups with 100 decryptors, corruption bound $c = 16$, and reconstruction threshold 17 gives mean computation per decryptor of 6.34 s for HSM-CL and 1.14 s for GJKR. These measurements include share and proof verification and key extraction. Complaint resolution and recovery are tested

$\Pi_{\mathsf{DSA}}.\mathsf{Step}_{5+3q}(Q^{r-1}, v_i^*, sk_i, sk_i^{r,5+3q})$

1. Put $s \coloneqq 5+3q$. Node i enters step s upon finishing step $s-1$ and waits at most 2δ. Hence, the maximum total time from T_i^r is:
$$\tau_s \coloneqq \tau_{s-1} + 2\delta.$$
During this wait, node i continuously checks the two ending conditions and processes step $(s-1)$ messages.
2. For each $s' \in [4, .. s-1]$, $b \in \{0,1\}$, and distinct v, let:
$$I_{b,v}^{r,s'} \coloneqq \left\{ j \in [n] : \begin{array}{c}(b_j, v_j) = (b, v) \in m_j^{r,s'} \\ \wedge \mathsf{MsgVf}^{r,s'}(m_j^{r,s'}) = 1\end{array} \right\}, I_b^{r,s'} \coloneqq \bigcup_v I_{b,v}^{r,s'}.$$
3. **Ending condition 0.** If, at any point, there exist $q' \in \mathbb{N}$, $u \coloneqq 5+3q' \leq s$, and:
$$v = (\mathsf{H}(B_\ell^r), \ell) \neq \perp \;\; s.t. \; |I_{0,v}^{r,u-1}| \geq t_{\mathrm{v}}^{r,u-1},$$
and node i has received the matching step 1 header:
$$(\mathrm{Head}(B_\ell^r), cred_\ell^{r,1}),$$
set $\mathsf{H}(B^r) \coloneqq \mathsf{H}(B_\ell^r)$ and:
$$CERT^r \coloneqq ((\mathrm{Head}(B_\ell^r), cred_\ell^{r,1}), \{m_j^{r,u-1}\}_{j \in I_{0,v}^{r,u-1}}).^a$$
Stop waiting, end round r, and return.
4. **Ending condition 1.** If, at any point, there exist $q' \in \mathbb{N}$ and $u \coloneqq 6+3q' \leq s$ *s.t.*:
$$|I_1^{r,u-1}| \geq t_{\mathrm{v}}^{r,u-1},$$
set $B^r \coloneqq B_\epsilon^r$ and:
$$CERT^r \coloneqq (\mathrm{Head}(B_\epsilon^r), \{m_j^{r,u-1}\}_{j \in I_1^{r,u-1}}).$$
Stop waiting, end round r, and return.
5. If **neither ending condition occurs** and, before τ_s ends:
$$|I_1^{r,s-1}| \geq t_{\mathrm{v}}^{r,s-1},$$
stop waiting and set $b_i \coloneqq 1$. Otherwise, when τ_s ends, set $b_i \coloneqq 0$.
6. Compute $cred_i^{r,s}$. If $i \notin SV^{r,s}$, return $\perp$. Otherwise, set:
$$m_i^{r,s} \coloneqq (\mathrm{ESIG}_i(b_i), \mathrm{ESIG}_i(v_i^*), cred_i^{r,s}),$$
destroy $sk_i^{r,s}$, and propagate $m_i^{r,s}$.

Fig. 17: The coin-fixed-to-0 BBA step for node $i \in [n]$ in step $s = 5+3q, q \in \mathbb{N}$ of round r.

separately through local fault injection. Shared parameter generation and communication are excluded, so the measurements do not represent distributed setup latency.

Willow. Willow's full setup shares the randomness used to generate each decryptor's AHE and auxiliary encryption keys [8]. Delivery reconstructs the auxiliary secret key, so this key and its associated sharing must be refreshed before another execution. Dropout recovery also reveals missing decryptors' AHE secret keys, constraining reuse of those contributions. Although Willow discusses amortizing AHE setup, the complete setup cannot be treated as an unrestricted one-time cost.

Our extension uses 59-out-of-99 sharing of each decryptor's key generation seeds, following Willow's t-out-of-$(m-1)$ specification for $m = 100$ decryptors and corruption bound $c = 16$. We retain the upstream AHE and KAHE

$\underline{\Pi_{\mathsf{DSA}}.\mathsf{Step}_{6+3q}(Q^{r-1}, v_i^*, sk_i, sk_i^{r,6+3q})}$

1. Put $s \coloneqq 6+3q$. Node i enters step s upon finishing step $s-1$ and waits at most 2δ. Hence, the maximum total time from T_i^r is:
$$\tau_s \coloneqq \tau_{s-1} + 2\delta.$$
During this wait, node i continuously checks the two ending conditions and processes step $(s-1)$ messages.
2. For each $s' \in [4, .. s-1]$, $b \in \{0,1\}$, and distinct v, let:
$$I_{b,v}^{r,s'} \coloneqq \left\{ j \in [n] : \begin{array}{c} (b_j, v_j) = (b, v) \in m_j^{r,s'} \\ \wedge \mathsf{MsgVf}^{r,s'}(m_j^{r,s'}) = 1 \end{array} \right\}, I_b^{r,s'} \coloneqq \bigcup_v I_{b,v}^{r,s'}.$$
3. Ending condition 0 and 1 are identical to those specified for step $5+3q$ in Fig. 17.
4. If **neither ending condition occurs** and, before τ_s ends:
$$|I_0^{r,s-1}| \geq t_{\mathrm{v}}^{r,s-1},$$
stop waiting and set $b_i \coloneqq 0$. Otherwise, when τ_s ends, set $b_i \coloneqq 1$.
5. Compute $cred_i^{r,s}$. If $i \notin SV^{r,s}$, return $\perp$. Otherwise, set:
$$m_i^{r,s} \coloneqq (\mathrm{ESIG}_i(b_i), \mathrm{ESIG}_i(v_i^*), cred_i^{r,s}),$$
destroy $sk_i^{r,s}$, and propagate $m_i^{r,s}$.

Fig. 18: The coin-fixed-to-1 BBA step for node $i \in [n]$ in step $s = 6+3q, q \in \mathbb{N}$ of round r.

cryptographic parameters and proof relations, adjusting KAHE packing to accommodate the auxiliary masks. The upstream partial decryption proof does not bind its witness secret to the decryptor's registered public key; our extension retains this limitation. Measurements cover all 100 honest decryptors and include auxiliary-key reconstruction, per-client seed decryption, and PRG mask removal. Each run performs fresh local setup; integrated dropout recovery and setup reuse or refresh are not evaluated.

OPA. Our OPA implementation generates independent client authentication and helper encryption key pairs, expands the public LWR matrix, and precomputes the bases for packed secret sharing. No DKG or trusted dealer is required. For 1,000 clients and 100 helpers, total local initialization averages 1.49 s over three runs, with the public matrix and sharing bases prepared once per run. This cost is measured separately from client submission. Committee selection, authenticated key distribution, and network communication are not measured.

F.2 PQ Proof Configurations and Coverage

Profiles and proof configurations. All profiles use three decryptors, reconstruction threshold two, and corruption bound one.

LNP22/LaZer [43,44] jointly proves two BFV encryption openings under distinct keys, checking bounded randomness and messages, zero padding, and protocol context binding. The relation contains 44 binary polynomial columns and 12 polynomial equations. The second plaintext is illustrative rather than a full-length hash seed. We use a portable scalar ARM build of LaZer, so its timings do not characterize the upstream optimized x86 implementation.

$\underline{\Pi_{\mathsf{DSA}}.\mathsf{Step}_{7+3q}(Q^{r-1}, v_i^*, sk_i, sk_i^{r,7+3q})}$

1. Put $s \coloneqq 7+3q$. Node i enters step s upon finishing step $s-1$ and waits at most 2δ. Hence, the maximum total time from T_i^r is:
$$\tau_s \coloneqq \tau_{s-1} + 2\delta.$$
During this wait, node i continuously checks the two ending conditions and processes step $(s-1)$ messages.
2. For each $s' \in [4, .. s-1]$, $b \in \{0,1\}$, and distinct v, let:
$$I_{b,v}^{r,s'} \coloneqq \left\{ j \in [n] : \begin{array}{c} (b_j, v_j) = (b, v) \in m_j^{r,s'} \\ \wedge \mathsf{MsgVf}^{r,s'}(m_j^{r,s'}) = 1 \end{array} \right\}, I_b^{r,s'} \coloneqq \bigcup_v I_{b,v}^{r,s'}.$$
3. Ending condition 0 and 1 are identical to those specified for step $5+3q$ in Fig. 17.
4. If **neither ending condition occurs** and, before τ_s ends:
$$|I_0^{r,s-1}| \geq t_{\mathrm{v}}^{r,s-1},$$
stop waiting and set $b_i \coloneqq 0$.
5. Otherwise, if, before τ_s ends:
$$|I_1^{r,s-1}| \geq t_{\mathrm{v}}^{r,s-1},$$
stop waiting and set $b_i \coloneqq 1$.
6. Otherwise, when τ_s ends, let:
$$I^{r,s-1} \coloneqq I_0^{r,s-1} \cup I_1^{r,s-1},$$
and compute:
$$\ell \coloneqq \underset{j \in I^{r,s-1}}{\arg\min} \mathsf{H}(cred_j^{r,s-1}),\ b_i \coloneqq \mathrm{lsb}(\mathsf{H}(cred_\ell^{r,s-1}, r)).$$
7. Compute $cred_i^{r,s}$. If $i \notin SV^{r,s}$, return $\perp$. Otherwise, set:
$$m_i^{r,s} \coloneqq (\mathrm{ESIG}_i(b_i), \mathrm{ESIG}_i(v_i^*), cred_i^{r,s}),$$
destroy $sk_i^{r,s}$, and propagate $m_i^{r,s}$.

Fig. 19: The coin-genuinely-flipped BBA step for node $i \in [n]$ in step $s = 7 + 3q, q \in \mathbb{N}$ of round r.

Unruh-transformed ZKB++ [62,19] uses 438 repetitions, 32-byte seeds, and 48-byte digests. Its decryption circuit contains 804,749 Boolean gates, including 186,399 AND gates. Longfellow [31] uses arithmetic over a 256-bit prime field, with rate parameter 7 and 160 queries. Both prove the same decryption predicate, checking the share and PRF-seed commitments, deterministic flooding noise derivation, and the partial decryption equation.

To reduce circuit depth, the Longfellow encoding supplies intermediate SHAKE256 states as auxiliary private witnesses and checks their consistency with the full hash computation.

Preparation costs. Means cover three randomized proofs for the same public statement. Per-proof preparation averages 3.70 ms for LaZer and 17.22 ms for Longfellow, separately from proving and verification. Parameter generation and circuit compilation are excluded. The decryption circuits embed statement-specific public data, so compilation cannot generally be amortized across different statements.

F.3 Scalability Experiments

We vary the client count n and vector length ℓ over $\{100, 200, 400, 600, 800, 1000\}$, fixing the other dimension at 1,000 and using 100 decryptors per scheme. Each point averages three complete workload runs, using within-run averages for per-party costs. The shared $(n, \ell) = (1000, 1000)$ profile is measured once per repetition. The reconstruction thresholds and implementation coverage remain as specified in Tab. 1.

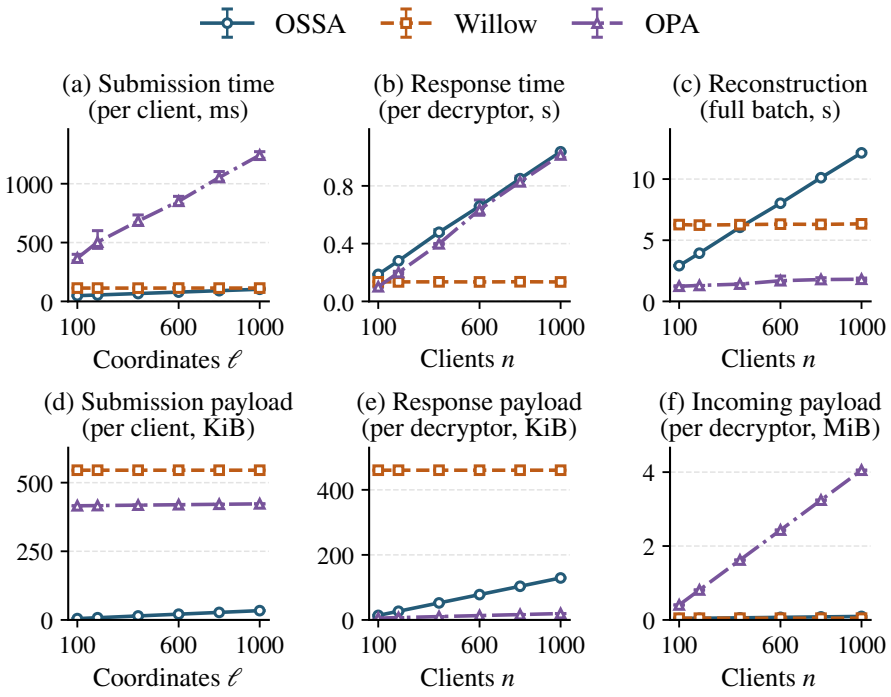

Fig. 20: Local computation and payload scaling. Panels (a,d) vary ℓ at $n = 1{,}000$; the remaining panels vary n at $\ell = 1{,}000$. Points show arithmetic means; whiskers show the observed minimum and maximum. Setup and client preparation are excluded from the plotted times. Payload accounting follows Tab. 1; panel (f) measures the payload received by each decryptor (helper in OPA).

Computation. As ℓ increases from 100 to 1,000, Ossa's classical submission time rises from 48.4 ms to 103 ms per client. Its decryptor response and reconstruction times grow approximately linearly with n, reaching 1.04 s per decryptor and 12.1 s per batch. Reconstruction verifies and combines threshold ElGamal shares for every client and regenerates *curve-valued* hash masks. OPA likewise exhibits increasing submission time with ℓ and decryptor response time with n. Willow's curves remain nearly flat over this range. Its auxiliary seed decryption and mask regeneration costs grow with n, but remain small relative to verification of the fixed 100 decryptor responses, which dominates reconstruction.

Payloads. Ossa's submission payload grows from 4.85 KiB to 33.9 KiB with ℓ, while its response payload grows from 13.9 KiB to 129 KiB with n, reflecting the partial decryptions and proofs for each client's hash-seed ciphertext. Willow's submission payload remains constant because all tested vectors fit in one fixed-size KAHE ciphertext. With 100 decryptors, its decryptor request and response sizes also remain constant. Verifier traffic grows with n and is not shown in these panels. OPA's submission payload changes little with ℓ, whereas its incoming payload per helper grows from 0.405 MiB to 4.05 MiB with n, reflecting the encrypted share envelopes received from all clients.

G Applications and Deployment Requirements

AlgOssa targets repeated aggregation services that tolerate failed rounds and variable delivery delays. Its eventual delivery guarantee matters when an unavailable or malicious aggregator could otherwise stall the service indefinitely.

Shared sensing and operational analytics. Consider infrastructure operators collecting private device reports on connectivity failures or unsuccessful software updates. Aggregate event counts support retrospective monitoring and maintenance planning [39,26]. Devices can retain observations for later rounds, so failed attempts delay statistics without necessarily making them useless.

With sufficient continued participation, later rounds can deliver a non-empty aggregate despite failed or withholding aggregators. The counts describe the finalized contributor set, not necessarily the full population. An empty round indicates that no statistic was delivered, not that no events occurred.

Jointly operated asynchronous learning. Consider organizations training a shared model whose clients complete updates at different times, motivating buffered aggregation [52]. If an aggregator fails or withholds its proposal, training can retain the current model until a later round aggregates eligible updates.

This integration requires tolerance for pauses between updates. Contributions must be bound to the training job and model version, with duplicate prevention and staleness control. Convergence and poisoning resistance require separate analysis.

Participation and retry requirements. Both scenarios require enough valid submissions to remain available to honest potential leaders, as specified in Sec. 4. Observations or updates can be buffered, but protocol submissions are bound to an execution and key epoch, so retries may require fresh submissions. Under the stated assumptions, AlgOssa guarantees eventual aggregate delivery, not inclusion of a particular contribution or completion by a fixed deadline.