

Continuum: Concretely Efficient Asynchronous Dynamic MPC with Guaranteed Output Delivery

Wenxuan Yu*, Minghui Xu^{*✉}, Yuan Lu[†], Yingzi Gao*, Xiuzhen Cheng*

^{*}Shandong University

[†]A*STAR Institute of Advanced Intelligence and Computing

haitengseat@gmail.com, mhxu@sdu.edu.cn, lu_yuan@a-star.edu.sg,

gaoyingzi@sdu.edu.cn, xzcheng@sdu.edu.cn

✉ Corresponding author

Abstract—Traditional secure multiparty computation (MPC) protocols assume a fixed set of participants throughout the computation. Dynamic MPC (DMPC) relaxes this assumption by permitting parties to join or leave during the process. However, practical deployment of DMPC in Internet settings favors tolerating asynchrony and achieving guaranteed output delivery (GOD). Recently, AD-MPC [CCS 2025] introduced an asynchronous DMPC (ADMPC) achieving this goal, but its reliance on per-epoch preprocessing and masking randomness incurs substantial concrete overhead. In this paper, we present Continuum, a concretely efficient ADMPC protocol with GOD. We introduce an aggregated hand-off protocol that ensures transfer consistency across epochs while eliminating per-epoch masking randomness, and a batch multiplication protocol based on the BGW paradigm that defers degree reduction to the hand-off phase and aggregates verification for all multiplications into succinct proofs. Together, these protocols enable a non-interactive computation phase with interaction concentrated in an efficient hand-off phase, overcoming the primary concrete inefficiencies of prior ADMPC designs. We implement Continuum and evaluate its performance on up to 128 geographically distributed servers. In experiments on 22 servers with circuits of depth 6 and width 100, Continuum reduces linear-gate transfer latency from 76.60s to 15.13s (80% reduction) and multiplication-gate evaluation time from 476.62s to 35.67s (93% reduction), compared with AD-MPC, while preserving strong security guarantees.

I. INTRODUCTION

Secure multiparty computation (MPC) [1]–[3] enables a set of mutually distrustful participants to jointly compute a function while keeping their respective inputs private. However, traditional MPC protocols assume a static set of participants who remain online throughout the entire computation, rendering them unsuitable for dynamic environments where servers may intermittently go offline due to network churn. To address this limitation, recent work has introduced *Dynamic MPC* (DMPC) [4], which partitions the computation into sequential epochs, each handled by a distinct set of participants, termed a *committee*. In a typical DMPC protocol, an epoch consists of two phases: a *computation phase*, during which the current

committee evaluates its assigned circuit segment, and a *hand-off phase*, where the resulting secret-shared state is securely transferred to the next committee. DMPC allows participants to dynamically join or leave across epochs, a crucial feature for long-running computations involving complex circuits, such as MPC-based shuffle [5], anonymous voting systems [6], and privacy-preserving machine learning [7].

DMPC naturally aligns with MPC-as-a-Service (MPCaaS), where the set of available servers changes over time. MPCaaS [5], [6] allows resource-constrained clients to outsource privacy-preserving computations to distributed servers, providing a practical deployment setting for DMPC. However, many MPCaaS systems are deployed over the Internet [5], where the message delay can be very large or even unpredictable. To maintain service reliability in such environments, it is essential to achieve *Guaranteed Output Delivery* (GOD), ensuring that all honest parties eventually obtain the correct output despite network instability and adversarial behavior. Therefore, reliable MPCaaS systems necessitate asynchronous DMPC (ADMPC)¹ protocols capable of achieving GOD. However, although numerous DMPC protocols have been proposed for synchronous networks [4], [9]–[16], their inherent reliance on strong synchrony assumptions precludes them from achieving GOD in asynchronous networks. This gap necessitates the development of concretely efficient ADMPC protocols that achieve GOD.

A. Obstacles to Concretely Efficient ADMPC with GOD

To achieve GOD in the asynchronous dynamic setting, an ADMPC protocol requires that every epoch executes correctly and eventually terminates, so that the overall evaluation can keep progressing and the final output is eventually delivered. This requires, in particular, that for every gate evaluated by committee $\mathcal{P}^l$ in epoch l , the resulting secret-shared output be correctly handed off to the next committee $\mathcal{P}^{l+1}$, preserving the underlying values for continued evaluation. The obstacles to achieving this efficiently differ between linear gates and multiplication gates.

¹ADMPC denotes a class of protocols. AD-MPC [8] refers to the specific instantiation of ADMPC.

a) *Obstacles for Linear Gate Evaluation:* Since linear gates are evaluated locally, the challenge lies entirely in transferring them correctly to the next committee, ensuring *transfer consistency*: the incoming committee $\mathcal{P}^{l+1}$ must interpolate a fresh degree- t sharing $[s]^{l+1}$ of the same secret encoded by the degree- t sharing $[s]^l$ held by $\mathcal{P}^l$, preserving the underlying secrets for continued evaluation. Most existing asynchronous MPC (AMPC) protocols [5], [17]–[23] are designed for the static setting and therefore provide no mechanism to transfer secret-shared state consistently across epochs. AD-MPC [8] addresses this gap through a mask-and-reconstruct strategy that guarantees transfer consistency in the asynchronous dynamic setting. However, achieving this guarantee incurs substantial concrete overhead, making it a primary bottleneck to concretely efficient ADMPC with GOD.

Obstacle #1: Masking randomness overhead in AD-MPC. To guarantee transfer consistency, AD-MPC [8] relies on a mask-and-reconstruct strategy, as illustrated in Figure 1. In epoch $l-1$, committee $\mathcal{P}^{l-1}$ distributes fresh masking randomness $[r]^l$ to committee $\mathcal{P}^l$. Upon receiving this randomness, $\mathcal{P}^l$ invokes the asynchronous dynamic transfer protocol ADtrans to mask each sharing $[s]^l$ and broadcast the masked values with correctness proofs, enabling committee $\mathcal{P}^{l+1}$ to interpolate fresh degree- t sharings $[s]^{l+1}$. The primary inefficiency of this strategy stems from the recurring requirement for masking randomness in every epoch. To transfer B sharings, the Randomness Generation step invokes n parallel Asynchronous Complete Secret Sharing (ACSS) instances [24], [25] and one Multi-valued Validated Byzantine Agreement (MVBA) [26], [27] to distribute masking randomness, incurring $O(\kappa n \log n \cdot B + \kappa n^3)$ communication per epoch, where κ denotes the security parameter. The subsequent ADtrans invocation likewise runs n parallel ACSS instances and one MVBA for the masked resharing, incurring $O(\kappa n^2 \log n \cdot B + \kappa n^3)$ communication. Although the communication cost associated with randomness generation is asymptotically smaller than that of ADtrans, it is unavoidable and recurs every epoch, further increasing concrete bandwidth overhead.

b) *Obstacles for Multiplication Gate Evaluation:* Multiplication gate evaluation requires interaction among committee members. For each multiplication gate in epoch l with input degree- t sharings $[a]^l$ and $[b]^l$, committee $\mathcal{P}^{l+1}$ must ultimately interpolate a fresh degree- t sharing $[c]^{l+1}$ encoding the exact product $c = a \cdot b$. Ensuring this demands that $\mathcal{P}^l$ faithfully evaluates the product and that the resulting output sharing is handed off to $\mathcal{P}^{l+1}$ with transfer consistency, which is a combined requirement we term *multiplication correctness*. In AMPC protocols, multiplication gates are typically evaluated using one of two paradigms: the offline/online paradigm based on Beaver triples [5], [17]–[21] or the BGW paradigm based on direct share multiplication [22], [23]. However, when these paradigms are applied in the asynchronous dynamic setting,

²Throughout the paper, square brackets denote degree- t sharings and angle brackets denote intermediate degree- $2t$ sharings; a subscript i denotes the local share held by server P_i , and a superscript l denotes the epoch.

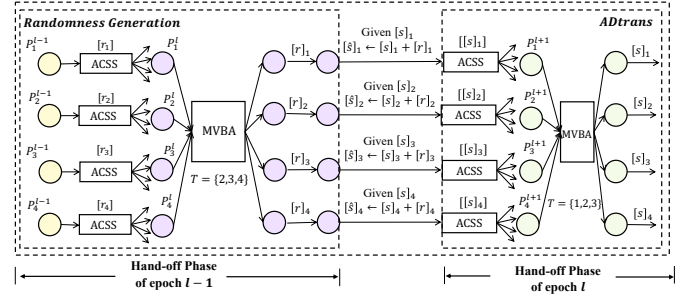


Fig. 1. Mask-and-reconstruct strategy in AD-MPC for transferring linear gate outputs. P_i^l denotes the i -th server of committee $\mathcal{P}^l$. The strategy proceeds in two steps: committee $\mathcal{P}^{l-1}$ distributes fresh masking randomness $[r]^l$ to $\mathcal{P}^l$; then $\mathcal{P}^l$ invokes ADtrans to mask sharing $[s]^l$ with $[r]^l$ and broadcasts the masked values, enabling $\mathcal{P}^{l+1}$ to interpolate fresh degree- t sharings $[s]^{l+1}$.

both encounter inherent obstacles to ensuring multiplication correctness efficiently.

Obstacle #2: Loss of preprocessing amortization in the offline/online paradigm for ADMPC. Existing ADMPC research has primarily leveraged the offline/online paradigm. To the best of our knowledge, AD-MPC [8] is the first ADMPC protocol to achieve GOD and adopts the offline/online paradigm. In this construction, the active committee runs an asynchronous dynamic preprocessing protocol ADprep to generate Beaver triples for the next epoch, which are then consumed during the computation phase to evaluate multiplication gates. Given correct Beaver triples, the computation phase guarantees online multiplication correctness via the robustness of Shamir secret sharing. To hand off the resulting output sharings to $\mathcal{P}^{l+1}$ with transfer consistency, AD-MPC then invokes ADtrans, as illustrated in Figure 2(a). Concretely, for M multiplication gates per epoch, guaranteeing online multiplication correctness requires $O(\kappa n^2 \log n \cdot M + \kappa n^3)$ communication for ADprep and $O(\kappa n \cdot M)$ for the online evaluation; transfer consistency of the output sharings via ADtrans further incurs $O(\kappa n^2 \log n \cdot M + \kappa n^3)$ communication, plus $O(\kappa n \log n \cdot M + \kappa n^3)$ for the associated randomness generation.

In static AMPC, the offline/online paradigm is highly effective because the server set is fixed, allowing parties to run one global preprocessing in advance to generate all required Beaver triples and distribute them to the servers, leaving the online evaluation lightweight. In ADMPC, however, committees are formed on the fly as the evaluation progresses; thus, one cannot run preprocessing once and distribute triples ahead of time for future committees that have not yet been formed. Consequently, ADprep must be invoked separately for each epoch, in lockstep with circuit evaluation. For a circuit whose multiplication gates are distributed across d epochs, this results in $O(d)$ independent preprocessing, compared to the static setting which requires only a single global preprocessing run in advance for all triples, eliminating the amortization benefit that underlies the concrete efficiency of the offline/online paradigm.

Obstacle #3: Absence of a compatible hand-off mecha-

nism in the BGW paradigm for ADMPC. To mitigate the overhead of epoch-by-epoch preprocessing, one alternative is to evaluate multiplication gates directly on secret-shared inputs using the BGW paradigm [3]. Recent AMPC protocols [22], [23] have adapted BGW-style multiplication to asynchronous networks. In these protocols, parties locally multiply their shares to obtain a degree- $2t$ sharing of the product, and then invoke an asynchronous verifiable degree-reduction procedure to interpolate a standard degree- t sharing, as illustrated in Figure 2(b). Since BGW-style multiplication avoids the need for per-epoch preprocessing, it can be executed directly in the active committee without a separate preprocessing pipeline. Compared to the offline/online paradigm, the BGW paradigm offers concrete communication improvements in the dynamic setting. Specifically, each dealer invokes only a single ACSS instance augmented with a constant-size proof, whereas the offline/online paradigm requires three ACSS instances per dealer for Beaver triple distribution, in addition to further online communication for consuming the triples.

However, applying the BGW paradigm to the asynchronous dynamic setting introduces two concrete obstacles. First, existing BGW-style AMPC protocols are designed for the static setting and provide no hand-off mechanism to transfer the degree- t output sharings to $\mathcal{P}^{l+1}$. While BGW-style degree reduction ensures that the computation phase produces degree- t sharings of the products within $\mathcal{P}^l$, no existing mechanism transfers these sharings to $\mathcal{P}^{l+1}$ with transfer consistency. Second, the degree-reduction procedure requires each dealer to broadcast product-relation proofs for the reshared products. In Dumbo-MPC [23], the current state-of-the-art BGW-style AMPC³, these proofs are instantiated over KZG polynomial commitments [28], resulting in $O(M)$ standalone product-relation proofs per dealer. Although the overall communication complexity remains $O(\kappa n^2 \cdot M + \kappa n^3)$, this linear product-relation argument component becomes significant at scale.

B. Our Contributions

In this work, we present Continuum, a concretely efficient ADMPC protocol achieving guaranteed output delivery. To address the obstacles above, Continuum introduces two novel protocols: AggTrans, an aggregated hand-off protocol that ensures transfer consistency while eliminating per-epoch masking randomness, and BatchMul, a BGW-based batch multiplication protocol that achieves multiplication correctness via a non-interactive computation phase. Table I compares their per-epoch communication complexities with those of AD-MPC. Together, these protocols yield a per-epoch communication complexity of $O(\kappa n^2 \cdot W + \kappa n^3)$ for circuit width $W = B + M$, where B and M denote the numbers of linear-gate and multiplication-gate output sharings per epoch, respectively. In sum, our contributions are:

³Dumbo-MPC is an offline/online AMPC protocol that uses BGW-style multiplication to generate Beaver triples; this same BGW multiplication procedure can be applied directly in the computation phase as a standalone degree-reduction step.

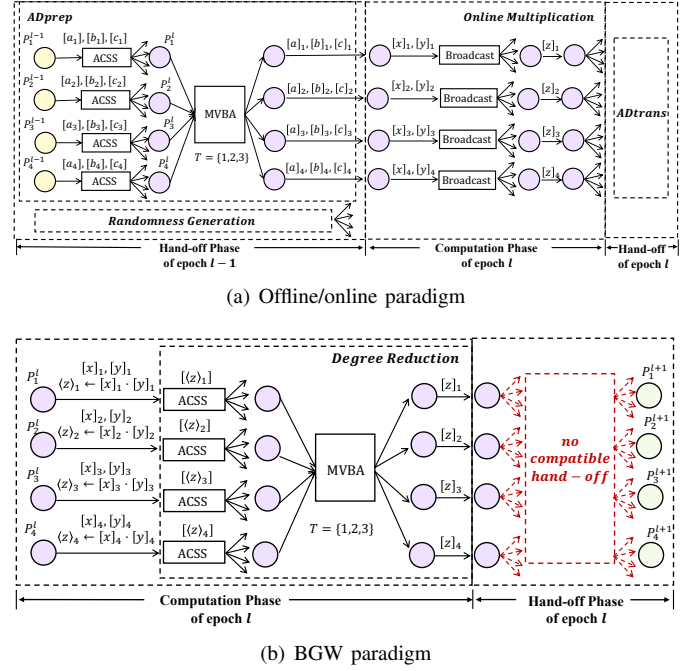


Fig. 2. Two paradigms for evaluating multiplication gates in the asynchronous dynamic setting. The red dashed region in (b) highlights the absence of a compatible hand-off mechanism in existing BGW-style AMPC protocols.

- **Aggregated hand-off protocol with commitment-anchored consistency.** We present AggTrans, a novel aggregated hand-off protocol that guarantees transfer consistency while eliminating per-epoch masking randomness. The key idea is a publicly verifiable anchoring condition: each dealer's resharing must prove that its constant term equals the dealer's own share of the gate output sharing committed to by the prior-epoch commitment. To realize this, we introduce *publicly verifiable aggregation interfaces* over KZG commitments [28], enabling the receiving committee to check the anchoring condition directly from the public transcript and agree on a common subset of valid senders. This removes the randomness-generation pipeline of AD-MPC, including n parallel ACSS invocations and one MVBA per epoch. The interfaces additionally aggregate the evaluation proofs for all B transferred sharings into a single constant-size proof, reducing the pairing operations required for anchoring verification from $O(B)$ to $O(1)$. By instantiating the underlying ACSS with the augmented KZG commitments and using MVBA for subset agreement, AggTrans achieves per-epoch transfer communication $O(\kappa n^2 \cdot B + \kappa n^3)$.
- **BGW-based batch multiplication with deferred degree reduction.** We design BatchMul, a BGW-based batch multiplication protocol that achieves multiplication correctness via a non-interactive computation phase. The key insight is that verifiable degree reduction can be deferred to the hand-off phase: parties locally multiply their degree- t shares to produce degree- $2t$ intermediate

TABLE I
COMPARISON OF ADMPC PROTOCOLS

Protocol	Threshold	GOD	Adversary	Setup	Gates	Communication ¹	Prover Comp.	Verifier Comp.	Rounds ²
AD-MPC [8]	$t < n/3$	✓	R-adaptive ³	–	Linear	$O(\kappa n^2 \log n \cdot B + \kappa n^3)$	$O(Bn^2)$	$O(Bn \log n)$	$2R_{\text{ACSS}} + 2R_{\text{MVBA}}$
					Multiplication	$O(\kappa n^2 \log n \cdot M + \kappa n^3)$	$O(Mn^2)$	$O(Mn \log n)$	$2R_{\text{ACSS}} + 3R_{\text{MVBA}} + 5$
Continuum	$t < n/3$	✓	R-adaptive	KZG CRS	Linear	$O(\kappa n^2 \cdot B + \kappa n^3)$	$O(Bn^2)$	$O(Bn)$	$R_{\text{ACSS}} + R_{\text{MVBA}}$
					Multiplication	$O(\kappa n^2 \cdot M + \kappa n^3)$	$O(Mn^2)$	$O(Mn)$	$R_{\text{ACSS}} + R_{\text{MVBA}}$

¹Per epoch, Linear: hand-off of B linear-gate output sharings; Multiplication: evaluation and hand-off of M multiplication-gate output sharings.

² R_{ACSS} and R_{MVBA} denote the round complexities of one ACSS and one MVBA invocation, respectively; each additional broadcast outside these invocations is counted as one round.

³An R-adaptive adversary may adaptively corrupt up to t servers at the beginning of each epoch, provided that corrupting a server that participated in an earlier committee does not raise the number of corrupted servers in that committee above t .

sharings during the computation phase, while the hand-off reshapes these intermediate sharings, reduces their degree, and proves the required product relations. Compared with a sequential construction that first performs degree reduction in the computation phase and then invokes a separate hand-off protocol such as AggTrans, BatchMul merges both steps into a single hand-off, reducing two sequential executions of n parallel ACSS instances and one MVBA to one. By employing a tailored inner-product argument over augmented KZG commitments, we encode all M product relations in a single $O(\log M)$ -size argument per dealer, reducing the product-relation proof size from $O(M)$ in prior BGW-style AMPC [23] to $O(\log M)$. As a result, BatchMul achieves per-epoch communication $O(\kappa n^2 \cdot M + \kappa n^3)$. Moreover, our approach of proving product relations of Shamir secret shares over KZG commitments could be of independent interest, such as making offline/online AMPCaaS with fixed participants more efficient [23].

- **Formal proof and prototype implementation.** We provide a formal security analysis of Continuum, demonstrating resilience against adversaries corrupting up to $t < n/3$ servers, alongside a prototype implementation evaluated on up to 128 geographically distributed servers. In particular, when $n = 22$, Continuum reduces linear-gate transfer latency under all-linear circuits of depth $d = 6$ and width $w = 100$ from 76.60 s to 15.13 s (an 80% reduction), and multiplication-gate evaluation time under all-multiplication circuits of the same dimensions from 476.62 s to 35.67 s (a 93% reduction), compared with AD-MPC. Our implementation is open-sourced at <https://anonymous.4open.science/r/ad-mpc-pp-C456>.

II. RELATED WORK

DMPC in the synchronous setting was pioneered by Fluid MPC [4], which introduced the notion of maximal fluidity, allowing a party to exit the computation immediately after completing a single round of sending and receiving messages. Subsequent synchronous DMPC protocols preserved maximal fluidity while achieving additional properties, such as tolerating a dishonest majority [9], [10] or attaining linear

communication complexity [9], [29]. However, these protocols provide only Security with Abort (SwA), meaning that honest parties can detect misbehavior but are not guaranteed to obtain the output. A line of work has further explored achieving GOD in the synchronous setting. DDG⁺23 [16] attains GOD under maximal fluidity, offering one protocol with perfect security for $t < n/3$ and another with computational security for $t < n/2$. DKLN25 [30] strengthens these results by showing that GOD can also be achieved with statistical security for $t < n/2$. Despite these advances, all such protocols fundamentally rely on synchrony and therefore cannot tolerate the arbitrary message delays of asynchronous networks. To the best of our knowledge, AD-MPC [8] is the first ADMPC protocol to achieve GOD in the asynchronous setting without assuming maximal fluidity. While AD-MPC demonstrates the feasibility of ADMPC with GOD, its concrete efficiency motivates further improvements.

Another related line of work is the YOSO (You Only Speak Once) model for MPC [11]–[13], [15], [31]–[36], which also seeks maximal fluidity by ensuring that each committee speaks only once before being replaced. This is achieved by hiding a committee’s identity until after its single broadcast, and has been studied in both synchronous and asynchronous settings. Compared to DMPC, YOSO MPC operates under a different threat model and incurs additional overhead from the role assignment protocol required to anonymously select committee members. Due to these fundamental differences, we do not consider YOSO MPC directly comparable to our setting.

III. MODEL AND PRELIMINARIES

In this section, we formalize our system model and threat assumptions, define the security goals of ADMPC, and introduce the necessary asynchronous building blocks used in our construction.

A. Model and Design Goals

Participants and Trusted Setup. We denote by $\mathcal{S} = \{P_1, \dots, P_N\}$ a finite set of N participating servers. We assume the existence of a public-key infrastructure (PKI), where each server $P_i \in \mathcal{S}$ holds a public-private key pair (pk_i, sk_i) , and all public keys pk_i are publicly known. In addition, we assume that the common reference string (CRS)

$(g, g^\alpha, \dots, g^{\alpha^t}, h, h^\alpha, \dots, h^{\alpha^t}) \in \mathbb{G}^{2t+2}$ for the KZG polynomial commitment scheme is honestly generated and publicly available. This CRS is generated once during the MPCaaS system deployment via an asynchronous distributed protocol [37]. The protocol is executed by an initial committee to support a fixed maximum degree, which corresponds to the system's security threshold. The resulting CRS is reusable across arbitrary circuit evaluations within this degree bound.

Asynchrony. We assume an asynchronous network, featuring authenticated point-to-point channels among all parties. Under this asynchronous model, there are no timing constraints, and messages may experience arbitrary delays; however, all messages sent between honest parties must eventually be delivered [38].

Dynamism. We consider an ADMPC protocol consisting of three consecutive stages: input, execution, and output. During the input stage, clients submit their private inputs and computational tasks to an initial committee $\mathcal{P}^1 \subseteq \mathcal{S}$. The execution stage involves evaluating the submitted tasks, represented as a layered arithmetic circuit $\mathcal{C}$ with depth d , partitioned into epochs $\{\mathcal{C}^l\}_{l \in [d]}$. For each epoch $l \in [d]$, the active committee $\mathcal{P}^l \subset \mathcal{S}$ first evaluates the epoch $\mathcal{C}^l$ in the computation phase and, if $l < d$, subsequently transfers the resulting secret-shared outputs to the next committee $\mathcal{P}^{l+1}$ during the hand-off phase. Finally, during the output stage, the last committee $\mathcal{P}^d$ delivers the final computation results to the clients.

Committee Configuration. Each committee consists of n servers selected from the global server set $\mathcal{S}$ according to a committee selection procedure. For each $l \in [d-1]$, the next committee $\mathcal{P}^{l+1}$ is selected before the hand-off phase of epoch $\mathcal{C}^l$ begins [4]. Committee $\mathcal{P}^{l+1}$ participates in the protocol from the beginning of the hand-off phase of epoch $\mathcal{C}^l$ through the computation and hand-off phases of epoch $\mathcal{C}^{l+1}$. We impose no constraints on committee overlap; a server may participate in multiple epochs.

Threat Model. We follow the R-adaptive adversary model for DMPC [4], [8]. For each $l \in [d]$, throughout the participation of committee $\mathcal{P}^l$ in the protocol, a probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ may adaptively corrupt up to $t < n/3$ servers in $\mathcal{P}^l$. Once a server is corrupted, $\mathcal{A}$ obtains its entire past state and may impersonate it in future communications. If a server P_i that previously participated in epoch $\mathcal{C}^m$ with committee $\mathcal{P}^m$ ($m < l$) is corrupted while participating in committee $\mathcal{P}^l$, the corruption is permitted only if it does not increase the number of corrupted servers in $\mathcal{P}^m$ beyond the threshold t . In addition, the adversary may corrupt up to c out of k clients submitting inputs during the input stage. For simplicity, we assume $n = k$ and $t = c$.

Design Goals. We aim to construct a concretely efficient ADMPC protocol with guaranteed output delivery. Formally, we define our goal as follows:

Definition 1 (*Asynchronous Dynamic MPC with Guaranteed Output Delivery*). *In an asynchronous network where all messages sent between honest parties are eventually delivered, an*

asynchronous dynamic MPC protocol for a layered arithmetic circuit with depth d and committees of size n achieves guaranteed output delivery against an R -adaptive PPT adversary $\mathcal{A}$ that may adaptively corrupt up to $t < n/3$ servers in each committee, if the protocol satisfies the following properties:

Privacy: *For each epoch $l \in [d]$, the adversary $\mathcal{A}$ learns nothing beyond what is revealed by the corrupted servers and clients.*

Guaranteed Output Delivery (Robustness): *For each epoch $l \in [d]$, the protocol satisfies the following three properties:*

- *Termination.* *After all honest servers in $\mathcal{P}^l$ complete the computation phase of epoch $\mathcal{C}^l$ and invoke the hand-off phase, every honest server in $\mathcal{P}^{l+1}$ eventually completes the hand-off phase and outputs its new secret shares.*
- *Transfer Consistency.* *After the computation phase of epoch $\mathcal{C}^l$, for every degree- t sharing $[s]^l$ held by honest servers in $\mathcal{P}^l$, the hand-off phase ensures that honest servers in $\mathcal{P}^{l+1}$ interpolate a fresh degree- t sharing $[s]^{l+1}$ of the same underlying secret.*
- *Multiplication Correctness.* *For each multiplication gate in $\mathcal{C}^l$ with input degree- t sharings $[a]^l$ and $[b]^l$ held by honest servers in $\mathcal{P}^l$, the servers locally compute a degree- $2t$ intermediate sharing $[c]^l$. The hand-off phase ensures that honest servers in $\mathcal{P}^{l+1}$ interpolate a fresh degree- t sharing $[c]^{l+1}$ that encodes the exact product $c = a \cdot b$.*

B. Preliminaries

Batch Asynchronous Complete Secret Sharing. A batch asynchronous complete secret sharing (BACSS) protocol consists of two phases: Sharing and Reconstruction. During the sharing phase, a dealer shares B secrets $(s_1, \dots, s_B)$ using Share. During the reconstruction phase, parties use Rec to recover these secrets. BACSS guarantees the following properties under asynchronous conditions:

- *Correctness:* If the dealer is honest, each honest party P_i will eventually receive shares $([s_1]_i, \dots, [s_B]_i)$ during the sharing phase.
- *Secrecy:* If the dealer is honest and no honest parties have begun the reconstruction phase, then a computationally bounded adversary corrupting up to t parties gains no information about s_k for any $k \in [B]$, except with negligible probability.
- *Completeness:* If any honest party completes the sharing phase, then for $k \in [B]$, there exists a degree- t polynomial $f_k(\cdot)$ over $\mathbb{Z}_q[x]$ such that $f_k(0) = s'_k$, and all honest parties P_i will eventually receive their shares $[s'_k]_i = f_k(i)$. Additionally, when the dealer is honest, $s'_k = s_k$.

One state-of-the-art BACSS protocol is hbACSS [25], which provides a modular framework that can be instantiated with various polynomial commitment schemes. In AD-MPC [8], hbACSS is instantiated with hbPolyCom [25], resulting in a communication complexity of $O(\kappa n \log n \cdot B)$, while in our setting we use KZG polynomial commitments [28], achieving $O(\kappa n \cdot B)$ for sharing B secrets.

Multi-valued Validated Byzantine Agreement. A Multi-valued Validated Byzantine Agreement (MVBA) protocol [27] allows a set of parties, where each party provides an input value, to eventually agree on the same value that satisfies a predefined external predicate $P(m) : \{0, 1\}^{|m|} \rightarrow \{0, 1\}$, which is known to all parties. The MVBA protocol satisfies the following properties [39]:

- *Termination:* If every honest party inputs with an externally valid value, then every honest party outputs a value.
- *External-Validity:* If an honest party outputs a value m , then $P(m) = 1$.
- *Agreement:* All honest parties that terminate output the same value.

We employ the MVBA protocol from [23], [40], [41] as a building block, which has a communication complexity of $O(\kappa n^3)$.

IV. ASYNCHRONOUS DYNAMIC MPC

We now present Continuum, our concretely efficient ADMPC protocol with GOD. We begin by outlining its overall structure. The protocol operates over a layered arithmetic circuit $\mathcal{C}$ of depth d , where each layer corresponds to an epoch. Let $\mathcal{C}^l$ denote the l -th epoch, characterized by its width w^l , which consists of λ^l linear gates and μ^l multiplication gates. Each epoch is evaluated by a committee $\mathcal{P}^l$ consisting of $n = 3t + 1$ servers, of which at most t may be corrupted. We assume an initial client set responsible for providing secret inputs. The execution of Continuum proceeds in three stages:

- **Input Stage:** Clients distribute their private inputs to the first committee $\mathcal{P}^1$ using BACSS.
- **Execution Stage.** During each epoch $l \in [d]$, committee $\mathcal{P}^l$ evaluates the corresponding epoch $\mathcal{C}^l$ through two sequential phases, as illustrated in Figure 3. Throughout epoch l , each wire value s is represented by a degree- t sharing $[s]$ together with a public polynomial commitment C_s^4 . Each server P_i holds the local share $[s]_i$ along with an evaluation proof $w_{s,i}$ proving that $[s]_i$ lies on the degree- t polynomial committed to by C_s .

Computation Phase. Each server P_i locally evaluates all w gates without interaction. For the k -th linear gate ($k \in [\lambda]$), with input wires x_k and y_k and output wire z_k , P_i computes the output share $[z_k]_i$. The associated commitment C_{z_k} and evaluation proof $w_{z_k,i}$ are updated linearly from the input commitments C_{x_k} , C_{y_k} and the corresponding proofs $w_{x_k,i}$, $w_{y_k,i}$, respectively, following the homomorphic properties of the commitment scheme. For the k -th multiplication gate ($k \in [\mu]$), with input wires a_k , b_k and output wire c_k , each server P_i locally computes $\langle c_k \rangle_i \leftarrow [a_k]_i \cdot [b_k]_i$, yielding a degree- $2t$ sharing $\langle c_k \rangle$.

Hand-off Phase. The committee $\mathcal{P}^l$ transfers the secret-shared state to the next committee $\mathcal{P}^{l+1}$. Specifically, $\mathcal{P}^l$ invokes *AggTrans* to transfer degree- t sharings of linear

gate outputs to ensure transfer consistency. Concurrently, $\mathcal{P}^l$ invokes *BatchMul* to reshare and reduce the degree of intermediate sharings, guaranteeing multiplication correctness.

- **Output Stage.** After the final epoch, clients receive shares of the outputs from committee $\mathcal{P}^d$ via the hand-off phase. The clients then broadcast these shares and reconstruct the final results, leveraging the robustness of Shamir secret sharing.

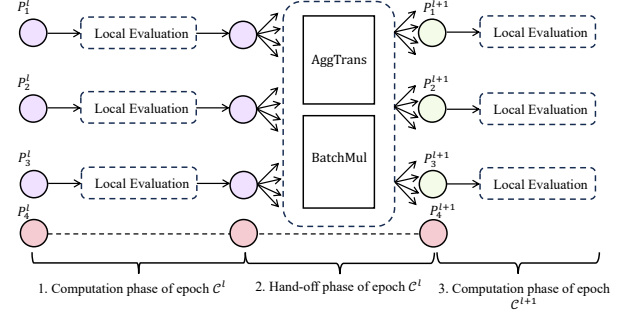


Fig. 3. Illustration of the execution stage in Continuum, including local evaluation, hand-off via *AggTrans* and *BatchMul*. Red nodes indicate corrupted parties.

Security and Overall Complexity. Full proofs and detailed complexity analysis for Continuum appear in Appendix E.

V. AGGREGATED HAND-OFF PROTOCOL

This section presents *AggTrans* by first explaining the masking-randomness bottleneck, then introducing the aggregation interfaces over KZG polynomial commitments, and finally describing the protocol construction.

A. Motivation: Eliminating the Masking Randomness Bottleneck

The hand-off phase must ensure transfer consistency across epochs. The core idea of AD-MPC's hand-off phase is that each server $P_i^l \in \mathcal{P}^l$ reshapes its local share $[s]_i^l$ as a degree- t sharing $[[s]_i^l]^{l+1}$ whose constant term is $[s]_i^l$. After agreeing on a common subset of dealers, $\mathcal{P}^{l+1}$ interpolates these resharing into a fresh degree- t sharing $[s]^{l+1}$. However, a malicious dealer may distribute a deviant resharing whose constant term differs from $[s]_i^l$, causing $\mathcal{P}^{l+1}$ to interpolate an incorrect sharing. AD-MPC prevents this inconsistency through its mask-and-reconstruct strategy: $\mathcal{P}^{l-1}$ first distributes fresh masking randomness to $\mathcal{P}^l$, which then invokes *ADtrans* to mask and broadcast its shares with correctness proofs. This recurring masking-randomness pipeline incurs substantial concrete overhead. This motivates a fundamental *anchoring condition*: every accepted resharing $[[s]_i^l]^{l+1}$ must be verifiably anchored to the prior commitment C_s^l , such that its constant term $[s]_i^l$ is a valid evaluation at the dealer's index i .

Our Approach. We achieve this condition by introducing an augmented KZG commitment scheme with publicly verifiable aggregation interfaces. Instead of relying on masking, each dealer provides a single constant-size, publicly verifiable proof

⁴For readability, within epoch l we omit the superscript l on local quantities (e.g., $[s]^l$, C_s^l , $w_{s,i}^l$) when the epoch is clear from context.

certifying that its reshared constant terms are valid evaluations of the commitments it broadcasts. This construction enables any server in the next committee $\mathcal{P}^{l+1}$ to independently verify the internal consistency of each dealer's resharing solely from the public transcript, without any access to original secret shares. However, these per-dealer public verifiability checks alone are insufficient to satisfy the anchoring condition. A malicious dealer could broadcast an arbitrary commitment $C_{s'}^l$ and provide a valid proof for a deviant share $[s']_i^l \neq [s]_i^l$, by-passing the above public verifiability checks. To defend against such attacks and ensure all accepted sharings are anchored to the original global commitment C_s^l , the next committee $\mathcal{P}^{l+1}$ invokes consensus to agree on a common subset of dealers who provide matching commitments that are identical to the globally agreed-upon commitments from the previous epoch. Interpolation over this agreed subset then interpolates fresh degree- t sharings of the original secrets, achieving transfer consistency without any masking randomness, as illustrated in Figure 4.

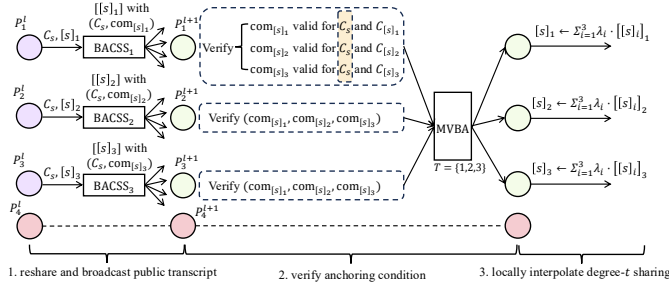


Fig. 4. Overview of AggTrans. C_s is the epoch- l KZG commitment to the gate output sharing, $C_{[s]_i}$ is the KZG commitment to P_i^l 's resharing polynomial, and $\text{com}_{[s]_i}$ is a Pedersen-style commitment to the reshared constant term broadcast together with a publicly verifiable evaluation proof. The yellow box highlights the anchoring condition, which requires all accepted dealers to broadcast the same C_s , ensuring their resharnings encode the correct epoch- l secret.

B. Aggregation Interfaces over KZG Polynomial Commitments

To satisfy the anchoring condition while reducing verification overhead, we augment the standard KZG polynomial commitment scheme [28] with the publicly verifiable aggregation interfaces in Figure 5. Following existing KZG batching techniques for multiple polynomial evaluations [42], we use a Fiat-Shamir random linear combination to aggregate the corresponding evaluation proofs, with challenge $\gamma = \mathcal{H}(SP, \text{ctx})$, where ctx is derived from the public transcript available before proof generation. When the claimed evaluations and their hiding randomness are available to the verifier, it can locally compute their aggregated values v_i and $\hat{v}_i$, and hence construct $T_i = g^{v_i} h^{\hat{v}_i}$. In our setting, the evaluations and hiding randomness must remain hidden from the verifier, and the dealer instead provides T_i . The aggregated KZG pairing check alone does not guarantee that the dealer knows an opening $(v_i, \hat{v}_i)$ of T_i . To enforce this knowledge condition, we employ a standard non-interactive zero-knowledge proof

of knowledge [43], [44] for the relation $T_i = g^{v_i} h^{\hat{v}_i}$, denoted by pok_{Ped} , with the following interfaces:

- $\text{pok}_{\text{Ped}}.\text{prove}(SP, \text{ctx}, T_i; v_i, \hat{v}_i)$: output a proof π_i^{ped} of knowledge of an opening $(v_i, \hat{v}_i)$ satisfying $T_i = g^{v_i} h^{\hat{v}_i}$.
- $\text{pok}_{\text{Ped}}.\text{verify}(SP, \text{ctx}, T_i, \pi_i^{\text{ped}})$: output 1 if π_i^{ped} is a valid proof of knowledge of an opening of T_i , and output 0 otherwise.

The resulting interfaces enable verification of the aggregated evaluations from the public transcript with $O(1)$ pairing operations, rather than the $O(B)$ pairing operations required to verify the B evaluation proofs individually.

KZG Polynomial Commitment with Aggregation

Setup($1^\kappa, t$): generate bilinear pairing group $\mathcal{G} = (e, \mathbb{G}, \mathbb{G}_T)$ and randomly sample $\alpha, \tau \in \mathbb{Z}_q$. Let g be random generator of $\mathbb{G}$, set $h = g^\tau$ and return $SP = \{\mathcal{G}, \{g^{\alpha^i}, h^{\alpha^i}\}_{i=0}^t\}$.

PolyCom($SP, f(\cdot)$): sample a random degree- t polynomial $\hat{f}(\cdot)$, compute $C = g^{f(\alpha)} h^{\hat{f}(\alpha)}$, and output $(C, \hat{f}(\cdot))$.

ProveEval($SP, i, f(\cdot), \hat{f}(\cdot)$): compute $w_i = g^{a_i(\alpha)} h^{\hat{q}_i(\alpha)}$, where $q_i(x) = \frac{f(x) - f(i)}{x - i}$ and $\hat{q}_i(x) = \frac{\hat{f}(x) - \hat{f}(i)}{x - i}$, output w_i .

VerifyEval($SP, C, i, f(i), \hat{f}(i), w_i$): output 1 if $e(C / (g^{f(i)} h^{\hat{f}(i)}), g) = e(w_i, g^\alpha / g^i)$, output 0 otherwise.

Below are publicly verifiable aggregation interfaces

AggPubProEval($SP, i, \text{ctx}, \{f_j(\cdot), \hat{f}_j(\cdot)\}_{j \in [B]}$): compute the challenge $\gamma = \mathcal{H}(SP, \text{ctx})$. Compute the aggregated polynomials:

$$h_i(x) = \sum_{j \in [B]} \gamma^{j-1} \cdot \frac{f_j(x) - f_j(i)}{x - i}, \quad \hat{h}_i(x) = \sum_{j \in [B]} \gamma^{j-1} \cdot \frac{\hat{f}_j(x) - \hat{f}_j(i)}{x - i}$$

Set $W_i = g^{h_i(\alpha)} h^{\hat{h}_i(\alpha)}$. Compute the aggregated evaluations:

$$v_i = \sum_{j \in [B]} \gamma^{j-1} f_j(i), \quad \hat{v}_i = \sum_{j \in [B]} \gamma^{j-1} \hat{f}_j(i).$$

Set $T_i = g^{v_i} h^{\hat{v}_i}$, and compute $\pi_i^{\text{ped}} = \text{pok}_{\text{Ped}}.\text{prove}(SP, \text{ctx}, T_i; v_i, \hat{v}_i)$. Output $(T_i, W_i, \pi_i^{\text{ped}})$.

AggPubVerEval($SP, i, \text{ctx}, T_i, W_i, \pi_i^{\text{ped}}, \mathbf{C}$): given the KZG commitment vector $\mathbf{C} = (C_1, \dots, C_B)$, recompute the challenge $\gamma = \mathcal{H}(SP, \text{ctx})$ and aggregate the commitments as $C = \prod_{j \in [B]} C_j^{\gamma^{j-1}}$. Output 1 if $\text{pok}_{\text{Ped}}.\text{verify}(SP, \text{ctx}, T_i, \pi_i^{\text{ped}}) = 1$ and $e(C/T_i, g) = e(W_i, g^\alpha / g^i)$; otherwise, output 0.

AggPedVerEval($SP, i, \text{ctx}, W_i, \text{Ped}, \mathbf{C}$): given the Pedersen-style evaluation-commitment vector $\text{Ped} = (\text{Ped}_1, \dots, \text{Ped}_B)$ and the KZG commitment vector $\mathbf{C} = (C_1, \dots, C_B)$, recompute the challenge $\gamma = \mathcal{H}(SP, \text{ctx})$ and aggregate

$$T_i = \prod_{j \in [B]} \text{Ped}_j^{\gamma^{j-1}}, \quad C = \prod_{j \in [B]} C_j^{\gamma^{j-1}}.$$

Output 1 if $e(C/T_i, g) = e(W_i, g^\alpha / g^i)$; otherwise, output 0.

Fig. 5. Our augmented KZG polynomial commitment with publicly verifiable aggregation interfaces. We describe its non-interactive variant using the Fiat-Shamir heuristic [45] in the random oracle model of the cryptographic hash function $\mathcal{H}$.

Security Properties. Our proposed KZG scheme with aggregation interfaces satisfies *correctness*, *public verifiability*, *evaluation binding*, *polynomial binding*, and *hiding*. These guarantees rely on the discrete logarithm and t -strong

Algorithm 1 Asynchronous dynamic transfer for $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$

Public Inputs: The batch size B , KZG commitment vector $\mathbf{C}^l = (C_1^l, \dots, C_B^l)$, and other system parameters

Private Inputs: P_i^l holds secret shares $\{[s_k]_i^l\}_{k \in [B]}$ with randomness $\{[\hat{s}_k]_i^l\}_{k \in [B]}$, evaluation proofs $\{w_{ki}^l\}_{k \in [B]}$

Public Outputs: New KZG commitment vector $\mathbf{C}^{l+1}$

Private Outputs: P_j^{l+1} obtains $\{[s_k]_j^{l+1}\}_{k \in [B]}$ with randomness $\{[\hat{s}_k]_j^{l+1}\}_{k \in [B]}$ and evaluation proofs $\{w_{kj}^{l+1}\}_{k \in [B]}$

Server P_i^l (in committee $\mathcal{P}^l$):

// 1. Reshare secrets using aggregation interfaces

```

101: for  $k \in [B]$ 
102:    $(C_{ki}^{l+1}, \hat{f}_{ki}^{l+1}(\cdot)) \leftarrow \text{PolyCom}(SP, f_{ki}^{l+1}(\cdot))$ ,
      // s.t.  $\hat{f}_{ki}^{l+1}(0) = [s_k]_i^l$ ,  $\hat{f}_{ki}^{l+1}(0) = [\hat{s}_k]_i^l$ 
103:   for  $j \in [n]$ 
104:      $w_{ki,j}^{l+1} \leftarrow \text{ProveEval}(SP, j, f_{ki}^{l+1}(\cdot), \hat{f}_{ki}^{l+1}(\cdot))$ 
105: let  $\mathbf{C}_i^{l+1}$  be the resulting KZG commitment vector and
      set  $\text{ctx}_i \leftarrow (\text{AggEval}, i, 0, \mathbf{C}^l, \mathbf{C}_i^{l+1})$ 
106:  $(T_{i,0}^{l+1}, W_{i,0}^{l+1}, \pi_i^{\text{ped}}) \leftarrow \text{AggPubProEval}(SP, 0, \text{ctx}_i,$ 
       $\{f_{ki}^{l+1}(\cdot), \hat{f}_{ki}^{l+1}(\cdot)\}_{k \in [B]})$ 
107: aggregate  $W_i^l \leftarrow \prod_{k \in [B]} (w_{ki}^l)^{\gamma_i^{k-1}}$  with the same chal-
      lenge  $\gamma_i \leftarrow \mathcal{H}(SP, \text{ctx}_i)$ 
108: invoke  $\text{BACSS\_share}_i$  as dealer to reshare  $\{[s_k]_i^l\}_{k \in [B]}$ 
      using KZG commitments  $\{C_{ki}^{l+1}\}_{k \in [B]}$  with broadcasting
       $(\mathbf{C}^l, W_i^l, W_{i,0}^{l+1}, T_{i,0}^{l+1}, \pi_i^{\text{ped}})$ 

```

Server P_j^{l+1} (in committee $\mathcal{P}^{l+1}$):

// 2. Locally verify shares and filter invalid dealers

```

201:  $T_j \leftarrow \emptyset$ ,  $\text{old\_shares}_j \leftarrow \emptyset$ ,  $\text{new\_shares}_j \leftarrow \emptyset$ 

```

```

202: activate  $\text{BACSS\_share}_i$  instance for every  $i \in [n]$ 
203: upon receiving  $\{C_{ki}^{l+1}, [[s_k]_{ij}^{l+1}], [[\hat{s}_k]_{ij}^{l+1}], w_{ki,j}^{l+1}\}_{k \in [B]}$ 
      and  $(\mathbf{C}^l, W_i^l, W_{i,0}^{l+1}, T_{i,0}^{l+1}, \pi_i^{\text{ped}})$  from  $\text{BACSS\_share}_i$ 
      // s.t.  $[[s_k]_{ij}^{l+1}] = f_{ki}^{l+1}(j)$ ,  $[[\hat{s}_k]_{ij}^{l+1}] = \hat{f}_{ki}^{l+1}(j)$ 
204:  $\text{ctx}_i \leftarrow (\text{AggEval}, i, 0, \mathbf{C}^l, \mathbf{C}_i^{l+1})$ 
205: if  $\forall k \in [B]$ , it holds that
       $\text{VerifyEval}(SP, C_{ki}^{l+1}, j, [[s_k]_{ij}^{l+1}], [[\hat{s}_k]_{ij}^{l+1}], w_{ki,j}^{l+1}) = 1$ 
       $\wedge \text{AggPubVerEval}(SP, 0, \text{ctx}_i, T_{i,0}^{l+1}, W_{i,0}^{l+1}, \pi_i^{\text{ped}}, \mathbf{C}_i^{l+1}) = 1$ 
       $\wedge \text{AggPubVerEval}(SP, i, \text{ctx}_i, T_{i,0}^{l+1}, W_{i,0}^{l+1}, \pi_i^{\text{ped}}, \mathbf{C}_i^{l+1}) = 1$ 
206:    $\text{old\_shares}_j \leftarrow \text{old\_shares}_j \cup \{[[s_k]_{ij}^{l+1}]\}_{k \in [B]}$ 
207:    $T_j \leftarrow T_j \cup \{i\}$ 
      // 3. Agree on subset with matching commitments
208:   if  $|T_j| \geq 2t + 1$  and  $\exists T'_j \subseteq T_j$  with  $|T'_j| = 2t + 1$ 
      such that all commitments  $\mathbf{C}^l$  are identical
209:     restrict  $T_j$  to dealers matching  $T'_j$ 's commitments
210:     call MVBA using  $T'_j$  as input and  $T_j$  as state
      // MVBA's predicate checks if  $T_j \supseteq \text{input}$ 
211: wait for MVBA output  $T$ , w.l.o.g. let  $T = \{P_1^l, \dots, P_{2t+1}^l\}$ 
      // 4. Interpolate new secret shares
212: for  $k \in [B]$ 
213:    $[s_k]_j^{l+1} \leftarrow \sum_{i \in T} \lambda_i [[s_k]_{ij}^{l+1}]$ 
      //  $\lambda_i$  is the Lagrange coeff. for eval at 0
      // update the corresponding KZG commitments and
      randomness for the new shares
214:    $\text{new\_shares}_j \leftarrow \text{new\_shares}_j \cup \{[s_k]_j^{l+1}\}_{k \in [B]}$ 
215: return  $\text{new\_shares}_j$ 

```

Diffie–Hellman assumptions, together with the completeness, zero-knowledge, and knowledge soundness of pok_{ped} in the random-oracle model. We provide formal definitions and proofs of these properties in Appendix A.

C. Construction of the AggTrans Protocol

We now present the detailed construction of AggTrans, which fulfills the anchoring condition to ensure transfer consistency without per-epoch masking randomness. The procedure is summarized in Algorithm 1 and organized into four phases:

- 1) *Aggregated resharing (lines 101–108)*. Each server $P_i^l \in \mathcal{P}^l$ reshapes its shares $\{[s_k]_i^l\}_{k \in [B]}$ by distributing new degree- t polynomials $f_{ki}^{l+1}(x)$ whose $f_{ki}^{l+1}(0) = [s_k]_i^l$. Crucially, P_i^l leverages the AggPubProEval interface to generate a succinct evaluation proof $(T_{i,0}^{l+1}, W_{i,0}^{l+1}, \pi_i^{\text{ped}})$ for all B evaluations at point 0. This proof, alongside the prior-epoch commitment vector $\mathbf{C}^l$ and the compressed prior-epoch evaluation proof W_i^l , enables the next committee to satisfy the anchoring condition and eliminates the need for masking randomness.

- 2) *Verification of anchoring (lines 201–207)*. This phase utilizes the public verifiability of our augmented KZG scheme to locally verify the consistency of each dealer's resharing relative to the prior epoch. Each $P_j^{l+1} \in \mathcal{P}^{l+1}$ performs the following checks:

- **Reshared share validity:** For every $k \in [B]$, VerifyEval ensures $[[s_k]_{ij}^{l+1}]$ is a correct evaluation of the new degree- t polynomial $f_{ki}^{l+1}(x)$ at point j .
- **Internal consistency:** AggPubVerEval verifies the succinct evaluation proof $(T_{i,0}^{l+1}, W_{i,0}^{l+1}, \pi_i^{\text{ped}})$ for the new commitment vector $\mathbf{C}_i^{l+1}$ at point 0 under the challenge context ctx_i .
- **Local anchoring to prior epoch:** The same $T_{i,0}^{l+1}$, π_i^{ped} , and challenge context ctx_i are then reused by AggPubVerEval , together with W_i^l , to verify the prior-epoch commitment vector $\mathbf{C}^l$ at point i , thereby anchoring the resharing to the prior-epoch evaluations of P_i^l .

- 3) *Agreement on matching commitments (lines 208–211)*. While local verification filters out malformed resharings,

it cannot individually guarantee that a dealer's commitments match the globally agreed-upon commitments from the previous epoch. A malicious dealer could provide shares that are consistent with its own commitments but diverge from the true global commitments. To prevent this, each P_j^{l+1} identifies a candidate subset $T_j' \subseteq T_j$ of size $2t+1$ whose members provide the same prior-epoch commitment vector $\mathbf{C}^l$. P_j^{l+1} then invokes MVBA with T_j' as input to reach consensus on a common subset of valid senders T . This step ensures that all servers in the new committee agree on the same set of resharings that are correctly anchored to the original secrets.

- 4) *Interpolation of new secret shares (lines 212–215).* After consensus is reached on the valid dealer set T , each P_j^{l+1} interpolates its final secret shares $\{[s_k]_j^{l+1}\}_{k \in [B]}$ using Lagrange interpolation over the shares $\{[s_k]_i^{l+1}\}_{i \in T}$. This yields a fresh degree- t sharing of each secret across the new committee $\mathcal{P}^{l+1}$, successfully preserving transfer consistency throughout the epoch transition.

Security of AggTrans. The security properties of AggTrans, including *termination*, *transfer consistency*, and *secrecy*, are formally stated and proven in Appendix B.

Complexity of AggTrans. Executing $\Theta(n)$ BACSS instances in parallel to share B secrets yields $O(\kappa n^2 \cdot B)$ communication, while a single MVBA invocation adds $O(\kappa n^3)$. Thus, the overall complexity of AggTrans is $O(\kappa n^2 \cdot B + \kappa n^3)$.

VI. BATCH MULTIPLICATION EVALUATION

This section presents BatchMul by first explaining the motivation for deferring verifiable degree reduction to the hand-off phase, then introducing the inner-product argument over augmented KZG commitments, and finally describing the protocol construction.

A. Motivation: Deferring Degree Reduction to the Hand-off Phase

AD-MPC's epoch-coupled preprocessing eliminates the amortization benefits of the offline/online paradigm. The BGW paradigm avoids this preprocessing by evaluating multiplication gates directly on secret shares. Since both Dumbo-MPC's BGW-style multiplication [23] and AggTrans are built upon KZG polynomial commitments, a straightforward BGW-style ADMPC construction can invoke them sequentially: Dumbo-MPC's multiplication protocol first performs verifiable degree reduction during the computation phase to produce degree- t product sharings, after which AggTrans transfers these sharings to $\mathcal{P}^{l+1}$ with transfer consistency. However, this construction incurs two concrete overheads. First, verifiable degree reduction requires $O(M)$ standalone product-relation proofs per dealer. Second, it executes n parallel BACSS instances and one MVBA for degree reduction, followed by another n parallel BACSS instances and one MVBA for hand-off, resulting in $2R_{\text{ACSS}} + 2R_{\text{MVBA}}$ rounds per epoch.

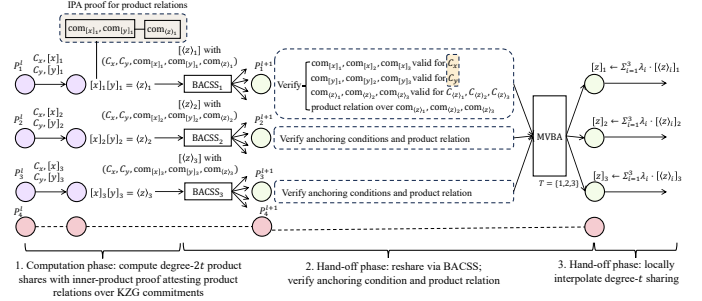


Fig. 6. Overview of BatchMul (exemplified with a single multiplication gate). $\langle z \rangle_i$ denotes the degree- $2t$ product share locally computed by P_i^l , and $C_{\langle z \rangle_i}$ is the KZG commitment to P_i^l 's resharing polynomial for the product.

Our Approach. We adopt the BGW paradigm while introducing two key technical optimizations to resolve the concrete inefficiencies inherent in the above construction. To reduce the linear product-relation argument component, we leverage our augmented KZG commitment together with a tailored inner-product argument that encodes all M reshared product relations in a single $O(\log M)$ -size argument per dealer. To resolve the redundant interaction bottleneck, we extend the commitment-anchored design of AggTrans to multiplication gates by deferring the verifiable degree-reduction procedure to the hand-off phase. Specifically, the computation phase becomes non-interactive, as each server in $\mathcal{P}^l$ locally multiplies its degree- t shares to obtain degree- $2t$ shares of the products. In the hand-off phase, each server in $\mathcal{P}^l$ reshapes these products to $\mathcal{P}^{l+1}$ by extending the aggregated hand-off protocol to simultaneously satisfy the anchoring condition and prove multiplicative relations. This approach ensures multiplication correctness across epochs with a logarithmic-size product-relation argument and interaction concentrated in the hand-off phase per epoch, as illustrated in Figure 6.

B. Inner Product Argument over Augmented KZG Commitments

We introduce IPA_{KZG} to prove the coordinate-wise product relations required by BatchMul over evaluations of KZG-committed polynomials. As illustrated in Figure 7, for each $x \in \{a, b, c\}$, let $\mathbf{C}_x = (C_{x,k})_{k \in [M]}$ denote the KZG commitment vector for polynomials evaluated at a common point i_x , let $\mathbf{x} = (f_{x,k}(i_x))_{k \in [M]}$ denote the corresponding evaluation vector, and let $\mathbf{Ped}_x = (\text{Ped}_{x,k})_{k \in [M]}$ denote the Pedersen-style evaluation-commitment vector. The required coordinate-wise relations are $f_{a,k}(i_a) \cdot f_{b,k}(i_b) = f_{c,k}(i_c)$ for all $k \in [M]$, which we write as the Hadamard-product relation $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$, where $\circ$ denotes component-wise multiplication.

These committed values must also be consistent with their corresponding KZG polynomial evaluations. For each $x \in \{a, b, c\}$, let W_x denote the aggregated evaluation witness computed from $\{w_{x,k}\}_{k \in [M]}$ as specified in Figure 7. Accord-

ingly, IPAKZG attests to the following joint statement:

$$\text{IPAKZG} \left[\begin{array}{l} \{(f_{x_k}(i_x), \hat{f}_{x_k}(i_x), w_{x_k})\}_{x \in \{a,b,c\}, k \in [M]} : \\ \text{Ped}_{x,k} = g^{f_{x_k}(i_x)} h^{\hat{f}_{x_k}(i_x)}, \forall x \in \{a,b,c\}, k \in [M] \\ \wedge \mathbf{a} \circ \mathbf{b} = \mathbf{c} \\ \wedge \text{AggPedVerEval}(SP, i_x, \text{ctx}, W_x, \\ \{\text{Ped}_{x,k}, C_{x_k}\}_{k \in [M]}) = 1, \quad \forall x \in \{a,b,c\} \end{array} \right]$$

This statement combines two components. For the first, we use an argument π to show that Ped_a , Ped_b , and Ped_c open to evaluation vectors satisfying $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$. The second, enforced by AggPedVerEval , verifies that the same committed values are consistent with their corresponding KZG polynomial evaluations. We instantiate π using the arithmetic-circuit construction of Bulletproofs [46], which reduces the first component to an inner-product relation and yields an $O(\log M)$ -size argument. As established in Theorem 5 of [46], the resulting Bulletproof argument provides *perfect completeness*, *statistical zero-knowledge*, and *computational soundness under the discrete logarithm assumption*. The detailed instantiation is provided in Appendix C.

Inner Product Argument over KZG Commitments IPAKZG

The prover $\mathcal{P}$ and the verifier $\mathcal{V}$ both know the public statement consisting of SP and, for each $x \in \{a,b,c\}$, the KZG commitment vector $\mathbf{C}_x = (C_{x,k})_{k \in [M]}$ together with its corresponding evaluation point i_x . $\mathcal{P}$ has private witness $\{(f_{x_k}(i_x), \hat{f}_{x_k}(i_x), w_{x_k})\}_{x \in \{a,b,c\}, k \in [M]}$, s.t. $\forall x \in \{a,b,c\}$ and $k \in [M]$, $\text{VerifyEval}(SP, C_{x,k}, i_x, f_{x_k}(i_x), \hat{f}_{x_k}(i_x), w_{x_k}) = 1$.

// **Prover** $\mathcal{P}$

$\text{IPAKZG}.\mathcal{P}(\{C_{x_k}, i_x, f_{x_k}(i_x), \hat{f}_{x_k}(i_x), w_{x_k}\}_{x \in \{a,b,c\}, k \in [M]}):$

- 1) set Pedersen-style evaluation-commitment vectors $\text{Ped}_x = (g^{f_{x_k}(i_x)} h^{\hat{f}_{x_k}(i_x)})_{k \in [M]}$, $x \in \{a,b,c\}$; set $\text{ctx} = (\text{IPAKZG}, i_a, i_b, i_c, \mathbf{C}_a, \mathbf{C}_b, \mathbf{C}_c, \text{Ped}_a, \text{Ped}_b, \text{Ped}_c)$ and compute $\gamma = \mathcal{H}(SP, \text{ctx})$; then aggregate the corresponding evaluation witnesses as $W_x = \prod_{k \in [M]} (w_{x_k})^{\gamma^{k-1}}$.
- 2) invoke the inner product argument prover on $\{(f_{x_k}(i_x), \hat{f}_{x_k}(i_x))\}_{x \in \{a,b,c\}, k \in [M]}$ as witness, and obtain the proof π regarding the product relation of $(\text{Ped}_a, \text{Ped}_b, \text{Ped}_c)$;
- 3) output proof = $(\{\text{Ped}_x, W_x\}_{x \in \{a,b,c\}}, \pi)$.

// **Verifier** $\mathcal{V}$

$\text{IPAKZG}.\mathcal{V}(\{(C_{x_k}, i_x)\}_{x \in \{a,b,c\}, k \in [M]}, \text{proof}):$

- 1) parse proof as $(\{\text{Ped}_x, W_x\}_{x \in \{a,b,c\}}, \pi)$;
- 2) set $\text{ctx} = (\text{IPAKZG}, i_a, i_b, i_c, \mathbf{C}_a, \mathbf{C}_b, \mathbf{C}_c, \text{Ped}_a, \text{Ped}_b, \text{Ped}_c)$ and recompute the challenge $\gamma = \mathcal{H}(SP, \text{ctx})$;
- 3) verify $\text{AggPedVerEval}(SP, i_x, \text{ctx}, W_x, \text{Ped}_x, \mathbf{C}_x) = 1$ for each $x \in \{a,b,c\}$;
- 4) verify the inner product proof π regarding $(\text{Ped}_a, \text{Ped}_b, \text{Ped}_c)$, return 1 if all checks pass and 0 otherwise.

Fig. 7. Construction of IPAKZG for proving coordinate-wise product relations over evaluations of KZG-committed polynomials.

C. Construction of the BatchMul protocol

Building on our IPAKZG construction, we present BatchMul, a batch multiplication protocol that extends AggTrans's an-

choring condition to multiplication gates and defers verifiable degree reduction to the hand-off phase. It guarantees that all multiplication outputs are correctly computed and encoded as fresh degree- t sharings for the next committee. Algorithm 2 summarizes the protocol in four phases:

- 1) *Proof generation and resharing (lines 101–106)*. Each server $P_i^l \in \mathcal{P}^l$ starts with the degree- $2t$ product shares $\{\langle c_k \rangle_i^l\}_{k \in [M]}$ produced during the non-interactive computation phase. It generates degree- t resharing polynomials whose constant terms are these product shares and computes the corresponding KZG commitments and evaluation proofs. The server then invokes $\text{IPAKZG}.\mathcal{P}$ to obtain proof_i , which attests that $\langle c_k \rangle_i^l = [a_k]_i^l \cdot [b_k]_i^l$ for every $k \in [M]$ and that the corresponding evaluations are consistent with their KZG commitments. Finally, P_i^l invokes BACSS to distribute the resulting shares together with the new KZG commitments and proof_i .
- 2) *Verification via IPAKZG (lines 201–205)*. Upon receiving valid BACSS shares and proof_i from a dealer P_i^l , each server $P_j^{l+1} \in \mathcal{P}^{l+1}$ invokes $\text{IPAKZG}.\mathcal{V}$ to verify all M product relations and their consistency with the corresponding KZG commitments. If the verification succeeds, P_j^{l+1} adds i to its candidate set T_j . However, this local verification alone does not guarantee that the input commitments correspond to the prior-epoch committed state, so the protocol additionally requires agreement on matching commitments.
- 3) *Agreement on subset with matching commitments (lines 206–209)*. Each server P_j^{l+1} identifies a subset $T'_j \subseteq T_j$ of size $2t + 1$ whose dealers provide identical input commitment vectors $\mathbf{C}_a^l$ and $\mathbf{C}_b^l$. It then restricts T'_j to dealers with matching commitments and invokes MVBA with T'_j as input. The resulting common subset T contains only multiplication shares verified against the same input commitments, preventing corrupted dealers from introducing locally valid products unrelated to the prior-epoch committed state.
- 4) *Interpolation of new multiplication sharings (lines 210–213)*. After consensus is reached on a valid dealer set $T = \{P_1^l, \dots, P_{2t+1}^l\}$ via MVBA, each server P_j^{l+1} reconstructs its share of every multiplication result $[c_k]_j^{l+1}$ by performing Lagrange interpolation over the shares $\{[\langle c_k \rangle_i^l]_j^{l+1}\}_{i \in T}$ for all $k \in [M]$. This yields fresh degree- t sharings of the multiplication outputs across $\mathcal{P}^{l+1}$, ensuring that the results remain faithful to those computed in epoch $\mathcal{C}^l$ and thereby upholding the multiplication correctness.

Security of BatchMul. The security properties of BatchMul, including *termination*, *multiplication correctness*, and *secrecy*, are formally stated and proven in Appendix D.

Complexity of BatchMul. To reshare M multiplication outputs, $\Theta(n)$ BACSS instances are executed in parallel, each contributing $O(\kappa n \cdot M)$ communication, for a total of $O(\kappa n^2 \cdot M)$. Each dealer additionally sends an inner-product proof of size $O(\kappa \log M)$, adding $O(\kappa n \log M)$ overall. Finally,

Algorithm 2 Batch multiplication for $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$

Public Inputs: KZG commitment vectors $\mathbf{C}_a^l, \mathbf{C}_b^l$, and other system parameters

Private Inputs: P_i^l holds degree- t shares $\{[a_k]_i^l, [b_k]_i^l\}_{k \in [M]}$ with randomness $\{\hat{a}_k]_i^l, [\hat{b}_k]_i^l\}_{k \in [M]}$, evaluation proofs $\{w_{a_k i}^l, w_{b_k i}^l\}_{k \in [M]}$, and degree- $2t$ product shares $\{\langle c_k \rangle_i^l\}_{k \in [M]}$ with randomness $\{\langle \hat{c}_k \rangle_i^l\}_{k \in [M]}$

Public Outputs: New KZG commitment vector $\mathbf{C}_c^{l+1}$

Private Outputs: P_j^{l+1} obtains $\{[c_k]_j^{l+1}\}_{k \in [M]}$ with randomness $\{[\hat{c}_k]_j^{l+1}\}_{k \in [M]}$ and evaluation proofs $\{w_{c_k i}^{l+1}\}_{k \in [M]}$

Server P_i^l (in committee $\mathcal{P}^l$):

// 1. Evaluate multiplications via IPA_{KZG} and reshare

```

101: for  $k \in [M]$ 
102:    $(C_{c_k i}^{l+1}, f_{c_k i}^{l+1}(\cdot)) \leftarrow \text{PolyCom}(SP, f_{c_k i}^{l+1}(\cdot))$ 
      // s.t.  $f_{c_k i}^{l+1}(0) = \langle c_k \rangle_i^l, \hat{f}_{c_k i}^{l+1}(0) = \langle \hat{c}_k \rangle_i^l$ 
103:   for  $j \in [0, n]$ 
104:      $w_{c_k i, j}^{l+1} \leftarrow \text{ProveEval}(SP, j, f_{c_k i}^{l+1}(\cdot), \hat{f}_{c_k i}^{l+1}(\cdot))$ 
105:  $\text{proof}_i \leftarrow \text{IPA}_{\text{KZG}}. \mathcal{P} \left( \left\{ \begin{pmatrix} (C_{a_k}^l, i), [a_k]_i^l, [\hat{a}_k]_i^l, w_{a_k i}^l, \\ (C_{b_k}^l, i), [b_k]_i^l, [\hat{b}_k]_i^l, w_{b_k i}^l, \\ (C_{c_k i}^{l+1}, 0), \langle c_k \rangle_i^l, \langle \hat{c}_k \rangle_i^l, w_{c_k i, 0}^{l+1} \end{pmatrix} \right\}_{k \in [M]} \right)$ 
      for  $k \in [M]$ 
106: invoke BACSS_share $i$  as dealer to reshare  $\{\langle c_k \rangle_i^l\}_{k \in [M]}$ 
      using KZG commitments  $\{C_{c_k i}^{l+1}\}_{k \in [M]}$  with broadcasting
       $(\{C_{a_k}^l, C_{b_k}^l\}_{k \in [M]}, \text{proof}_i)$ 

```

Server P_j^{l+1} (in committee $\mathcal{P}^{l+1}$):

// 2. Verify multiplication correctness via IPA_{KZG}

201: $T_j \leftarrow \emptyset, \text{mul\_shares}_j \leftarrow \emptyset$

```

202: activate BACSS_share $i$  instance for every  $i \in [n]$ 
203: upon receiving valid shares  $\{([\langle c_k \rangle_i^{l+1}], [\langle \hat{c}_k \rangle_i^{l+1}], C_{c_k i}^{l+1}, w_{c_k i, j}^{l+1})\}_{k \in [M]}$  and  $(\{C_{a_k}^l, C_{b_k}^l\}_{k \in [M]}, \text{proof}_i)$  from BACSS_share $i$ 
      // Validity verified via VerifyEval
      // s.t.  $[\langle c_k \rangle_i^{l+1}]_j = f_{c_k i}^{l+1}(j), [\langle \hat{c}_k \rangle_i^{l+1}]_j = \hat{f}_{c_k i}^{l+1}(j)$ 
204:   if  $\text{IPA}_{\text{KZG}}. \mathcal{V} \left( \left\{ \begin{pmatrix} (C_{a_k}^l, i), \\ (C_{b_k}^l, i), \\ (C_{c_k i}^{l+1}, 0) \end{pmatrix} \right\}_{k \in [M]}, \text{proof}_i \right) = 1$ 
205:      $T_j \leftarrow T_j \cup \{i\}$ 
      // 3. Agree on subset with matching commitments
206:     if  $|T_j| \geq 2t+1$  and  $\exists T'_j \subseteq T_j$  with  $|T'_j| = 2t+1$ 
      such that the polynomial commitments  $C_{a_k}^l$  and  $C_{b_k}^l$  in  $T'_j$ 
      match for all  $k \in [M]$ 
207:       restrict  $T_j$  to dealers matching  $T'_j$ 's commitments
208:       call MVBA using  $T'_j$  as input and  $T_j$  as state
      // MVBA's predicate checks if  $T_j \supseteq \text{input}$ 
209: wait for MVBA output  $T$ , w.l.o.g, let  $T = \{P_1^l, \dots, P_{2t+1}^l\}$ 
      // 4. Interpolate new secret shares
210: for  $k \in [M]$ 
211:    $[c_k]_j^{l+1} \leftarrow \sum_{i \in T} \lambda_i [\langle c_k \rangle_i^l]_j^{l+1}$ 
      //  $\lambda_i$  is the Lagrange coeff. for eval at 0
      // update the corresponding KZG commitments and
      randomness for the new shares
212:    $\text{mul\_shares}_j \leftarrow \text{mul\_shares}_j \cup \{[c_k]_j^{l+1}\}_{k \in [M]}$ 
213: return  $\text{mul\_shares}_j$ 

```

a single MVBA agreement incurs $O(\kappa n^3)$ communication. Thus, the total complexity of BatchMul is $O(\kappa n^2 \cdot M + \kappa n^3)$.

VII. IMPLEMENTATION AND EVALUATION

In this section, we implement a prototype of Continuum and evaluate its performance through system setup, sub-protocol benchmarks, and comparisons with AD-MPC and Dumbo-MPC.

A. Implementation and Evaluation Setup

Implementation. We have implemented a prototype of Continuum. The core protocol components, including AggTrans and BatchMul, are implemented in Python 3 as forks of the open-source implementations of AD-MPC [8] and Dumbo-MPC [23]. The authenticated point-to-point channels are instantiated using ZeroMQ CURVE over TCP.⁵ Concurrent tasks are managed via Python's `asyncio`⁶ library. Continuum relies on the pairing-friendly elliptic curve BLS12-381, instantiating the KZG polynomial commitment

⁵CURVE provides authentication, integrity, and confidentiality, whereas our model requires only authentication and integrity.

⁶<https://github.com/python/cpython>

scheme [28] with aggregation interfaces and the inner product argument [46] over augmented commitments. For the trusted setup, an initial committee executes the asynchronous distributed protocol [37] to generate the KZG CRS. For committee election, the current committee jointly derives a random beacon output. The beacon output seeds a deterministic permutation [47] to select the next committee; hence, all servers derive the same next committee from the same beacon output. All finite field operations are performed over a 256-bit prime field $\mathbb{Z}_q$ for Shamir secret sharing.

Evaluation Setup. Each committee comprises n servers and tolerates up to $t < n/3$ corruptions. The client set includes $k = n$ clients, of which up to $c = t$ may also be corrupted. To evaluate the efficiency and robustness of our design, we focus on four central questions: (i) the efficiency and scalability of AggTrans and its two design components; (ii) the efficiency and scalability of BatchMul relative to AD-MPC and BGW-AggTrans, which sequentially combines Dumbo-MPC's BGW-style multiplication [23] with AggTrans; (iii) GOD under accumulating corruptions, message delays, and malicious hand-off attacks, compared with AD-MPC and Dumbo-MPC; and (iv)

the end-to-end performance of a concrete MPC-based shuffle application benchmark relative to AD-MPC and static AMPC under different committee-change frequencies.

For questions (i) and (ii), we use rectangular circuits with depth $d = 6$ and width $w = 100$, consisting entirely of linear or multiplication gates, respectively, and evaluate committee sizes up to $n = 128$. For question (iii), we fix $n = 16$ and use mixed circuits with depth $d = 6$, width $w = 100$, and a 1:1 ratio of multiplication to linear gates. For question (iv), we evaluate a 128-input butterfly switching network [5] with 49 switch layers, one per epoch. We compare Continuum with committee changes every epoch, every five epochs, or not at all against AD-MPC and BGW-AMPC, a static baseline using Dumbo-MPC’s BGW-style multiplication [23]. Appendix F-A details the circuit construction and protocol adaptations. The sub-protocol evaluations for questions (i) and (ii) exclude committee election to isolate the latency of AggTrans and BatchMul. For all end-to-end evaluations, we add the independently measured committee-election latency between the computation and hand-off phases whenever the committee changes. Appendix F-B additionally evaluates the impact of circuit depth on the latency of Continuum, AD-MPC, and Dumbo-MPC.

We deploy a geographically distributed 128-node WAN testbed using AWS EC2 `c6i.2xlarge` instances, each equipped with 8 vCPUs and 16 GiB of memory and running Ubuntu 24.04. Each experiment is repeated twenty times, and we report the average latency with error bars spanning the observed minimum and maximum; some error bars are too narrow to be visible at the plotted scale.

B. Evaluation Results

System overheads. Authenticated point-to-point channels are established once and reused across circuit evaluations, with a maximum channel-establishment latency of 20.94 s at $n = 128$. Generating a reusable KZG CRS supporting polynomials of degree up to 63 takes at most 4.56 s at $n = 16$. These one-time costs are excluded from the reported circuit-evaluation latency, whereas committee election is incurred whenever the committee changes, with mean latencies of 76.03, 99.65, and 128.75 ms for $n = 4$, $n = 10$, and $n = 16$, respectively.

Transfer protocol evaluation. As shown in Figure 8, ADtrans dominates AD-MPC’s transfer latency, with additional overhead from RandGen, whereas AggTrans eliminates RandGen entirely. To isolate the contributions of our two design components, we include AggTrans-NoAgg as an ablation baseline. The commitment-anchored design alone achieves a 1.4–2.2 $\times$ speedup over AD-MPC by eliminating the masking-randomness pipeline and benefiting from the lower communication complexity of KZG over hbPolyCom [25], while reducing the communication per hand-off and the per-server local cryptographic overhead per circuit layer by up to 76.9% and 67.15%, respectively. The aggregation interfaces provide a further 2.0–4.3 $\times$ latency reduction by compressing the evaluation proofs for all B transferred sharings into a single constant-size aggregated proof, while further reducing the communication

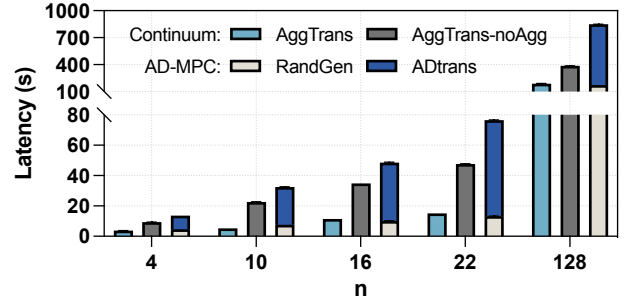


Fig. 8. Latency of transfer protocols under all-linear circuits ($d = 6$, $w = 100$). AD-MPC is decomposed into RandGen and ADtrans. AggTrans-NoAgg denotes AggTrans without aggregation interfaces, serving as an ablation to separate the contribution of the commitment-anchored design from the aggregation interfaces.

per hand-off and the per-server local cryptographic overhead per circuit layer by up to 31.03% and 74.4%, respectively. Together, AggTrans reduces the total transfer latency of AD-MPC by 3.7–6.2 $\times$. Detailed communication and computation breakdowns are provided in Appendix F-C.

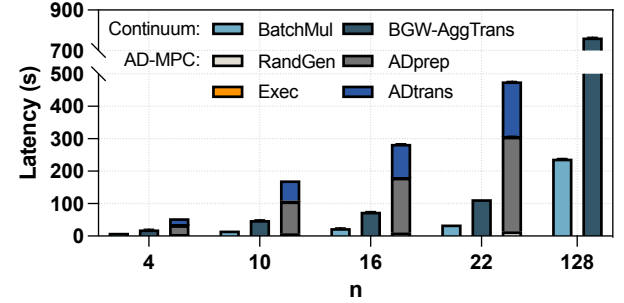


Fig. 9. Latency of multiplication protocols under all-multiplication circuits ($d = 6$, $w = 100$). AD-MPC is decomposed into ADprep, RandGen, Exec, and ADtrans. BGW-AggTrans denotes a sequential construction using Dumbo-MPC’s BGW-style multiplication [23] followed by AggTrans, serving as an ablation to isolate the contribution of each optimization. AD-MPC is not evaluated at $n = 128$ due to its substantially higher latency.

Multiplication protocol evaluation. As shown in Figure 9, the latency of AD-MPC consists of four components: ADprep for generating Beaver triples, RandGen for distributing masking randomness, Exec for consuming triples to evaluate multiplication gates, and ADtrans for transferring secret-shared outputs to the next committee. To quantify the contribution of each optimization, we include BGW-AggTrans as an ablation baseline. Replacing AD-MPC’s offline/online paradigm with the sequential BGW-style construction already achieves a 2.6–4.2 $\times$ speedup by eliminating per-epoch preprocessing and masking overhead, while reducing the communication per multiplication layer and the per-server local cryptographic overhead per circuit layer by up to 89.27% and 82.51%, respectively. Our inner-product argument and deferred degree-reduction design provide a further 2.1–3.2 $\times$ latency reduction over BGW-AggTrans by encoding the M product relations in a single $O(\log M)$ -size argument per dealer and combining

degree reduction and resharing within a single hand-off, while further reducing the communication per multiplication layer and the per-server local cryptographic overhead per circuit layer by up to 48.32% and 72.31%, respectively. Together, BatchMul achieves a 5.5–13.4 $\times$ reduction in overall latency compared with AD-MPC. We do not evaluate AD-MPC at $n = 128$ because its latency already reaches 476.62 s at $n = 22$, nearly twice the 238.54 s latency of BatchMul at $n = 128$. Detailed communication and computation breakdowns are provided in Appendix F-D.

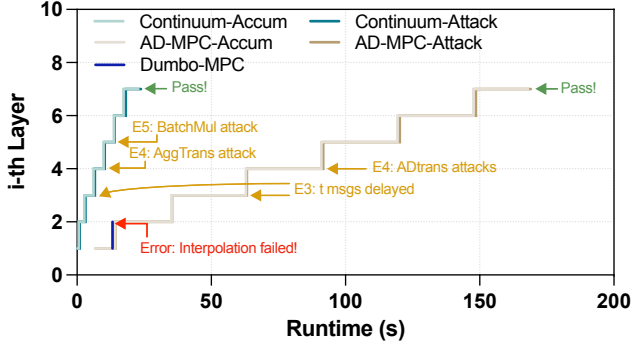


Fig. 10. Protocol progress on a mixed circuit ($d = 6$, $w = 100$, $n = 16$, $t = 5$). For the *Accum* curves, the adversary corrupts t previously uncorrupted servers in each epoch, causing corruptions to accumulate across epochs. The *Attack* curves additionally include message delays and malicious attacks during hand-off. Layer 7 is the output stage.

GOD under accumulating corruptions and Byzantine hand-off faults. We evaluate GOD under two adversarial settings. (i) Under *Accum*, the adversary corrupts t previously uncorrupted servers in each epoch, causing corruptions to accumulate across epochs. Continuum and AD-MPC use disjoint committees across epochs, whereas Dumbo-MPC reuses a single static committee. (ii) Under *Attack*, the adversary additionally introduces message delays and malicious attacks during hand-off. In epoch 3, it delays all hand-off messages from the t corrupted servers by 10 seconds. In epoch 4, the corrupted servers in Continuum use internally consistent BACSS instances in AggTrans to reshare values unrelated to the committed circuit state, with an analogous attack applied to ADtrans. In epoch 5, they use internally consistent BACSS instances in BatchMul with products derived from modified multiplication inputs.

As shown in Figure 10, under *Accum*, Dumbo-MPC’s static committee contains $2t$ corrupted servers by epoch 2, preventing it from completing that epoch. In contrast, every active committee of Continuum and AD-MPC contains at most the t servers corrupted in the corresponding epoch, allowing both protocols to progress through all epochs. Under *Attack*, Continuum and AD-MPC complete the hand-off in epoch 3 after receiving valid messages from $2t + 1$ servers, while excluding the t delayed servers from the common subset. In epochs 4 and 5, the adversarial BACSS instances in Continuum pass local verification but are excluded before MVBA because the t corrupted dealers cannot form the

required subset of $2t + 1$ dealers with identical prior-epoch commitment vectors in AggTrans or matching input commitments in BatchMul, respectively. AD-MPC similarly filters the malicious ADtrans instances. Consequently, both Continuum and AD-MPC achieve GOD, with end-to-end latencies of 23.84 s and 169.02 s, respectively, yielding a 7.1 $\times$ speedup for Continuum.

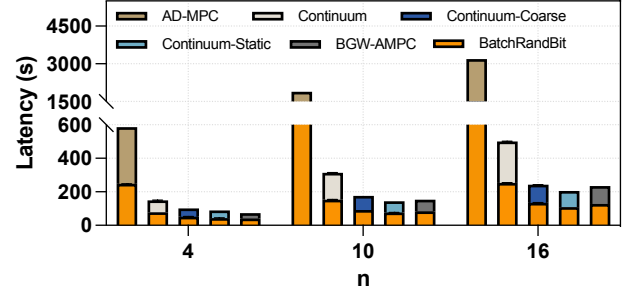


Fig. 11. End-to-end latency of the 128-input MPC-based shuffle over 49 switch layers, one per epoch. Continuum and AD-MPC change committees every epoch, Continuum-Coarse changes committees every five epochs, and Continuum-Static uses a fixed committee. BGW-AMPC denotes a static AMPC baseline using Dumbo-MPC’s BGW-style multiplication [23]. BatchRandBit denotes the generation of the secret-shared random sign bits required by each switch layer.

End-to-end MPC-based shuffle application benchmark.

To evaluate Continuum on a concrete MPC application, we instantiate a 128-input MPC-based shuffle using the butterfly switching network detailed in Appendix F-A. As shown in Figure 11, Continuum reduces end-to-end latency by 74.6%–84.3% relative to AD-MPC across the evaluated committee sizes. Reducing the committee-change frequency further lowers latency. At $n = 10$, the end-to-end latency decreases from 313.56 s for Continuum to 175.70 s for Continuum-Coarse and 143.45 s for Continuum-Static. Moreover, Continuum-Static outperforms BGW-AMPC when $n \geq 10$. Across the three Continuum configurations and all evaluated committee sizes, the BatchRandBit component accounts for 49.0%–55.6% of the end-to-end latency.

VIII. CONCLUSION AND FUTURE WORK

We presented Continuum, a concretely efficient asynchronous dynamic MPC protocol with guaranteed output delivery. By introducing AggTrans to eliminate per-epoch masking randomness and BatchMul for efficient multiplication gate evaluation, we address the concrete inefficiencies of prior designs. Our implementation demonstrates both strong security guarantees and substantial performance improvements in asynchronous dynamic environments.

The shuffle benchmark also reveals a limitation of applying Continuum to concrete applications. Although AggTrans and BatchMul reduce the generic ADMPC costs of state transfer and multiplication, concrete applications may require additional protocols that introduce substantial costs in asynchronous dynamic settings. For the shuffle benchmark,

BatchRandBit accounts for approximately half of the end-to-end latency. Designing efficient protocols for such application-specific requirements in asynchronous dynamic settings is an important direction for future work.

GENERATIVE AI USAGE

We used ChatGPT to assist with drafting and refining parts of the manuscript and to produce an initial code skeleton during early-stage implementation. All protocol designs, algorithms, security arguments, and experimental logic were developed by the authors. The AI-generated code drafts were manually reviewed, completed, and tested by the authors, and all experimental results were reproduced from the final, human-verified implementation.

ACKNOWLEDGMENTS

This study was supported by the National Natural Science Foundation of China (No. U25A20425, U23A20302, 62302266, 62232010, U24A20244), project ZR2026QA06 supported by Shandong Provincial Natural Science Foundation, Taishan Scholars Program under Grant tsqn202507069, Key technology research and industrialization demonstration projects in Qingdao 26-1-1-gjgg-30-gx and the Research Project of Quancheng Laboratory, China under Grant No. QCL20250106.

REFERENCES

- [1] A. C.-C. Yao, “How to generate and exchange secrets,” in *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 1986, pp. 162–167.
- [2] O. Goldreich, S. Micali, and A. Wigderson, “How to play any mental game, or a completeness theorem for protocols with honest majority,” in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.
- [3] M. Ben-Or, S. Goldwasser, and A. Wigderson, “Completeness theorems for non-cryptographic fault-tolerant distributed computation,” in *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 351–371.
- [4] A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk, “Fluid mpc: secure multiparty computation with dynamic participants,” in *Advances in Cryptology—CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part II 41*. Springer, 2021, pp. 94–123.
- [5] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller, “Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 887–903.
- [6] A. Barak, M. Hirt, L. Koskas, and Y. Lindell, “An end-to-end system for large scale p2p mpc-as-a-service and low-bandwidth mpc for weak participants,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 695–712.
- [7] Z. Liu, I. Tjuawinata, C. Xing, and K.-Y. Lam, “Mpc-enabled privacy-preserving neural network training against malicious attack,” *arXiv preprint arXiv:2007.12557*, 2020.
- [8] W. Yu, M. Xu, B. Wu, S. Duan, and X. Cheng, “Ad-mpc: Asynchronous dynamic mpc with guaranteed output delivery,” in *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, 2025, pp. 2892–2906.
- [9] A. Bienstock, D. Escudero, and A. Polychroniadou, “On linear communication complexity for (maximally) fluid mpc,” *Cryptology ePrint Archive*, 2023.
- [10] R. Rachuri and P. Scholl, “Le mans: Dynamic and fluid mpc for dishonest majority,” in *Annual International Cryptology Conference*. Springer, 2022, pp. 719–749.
- [11] C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakubov, “Yoso: You only speak once: Secure mpc with stateless ephemeral roles,” in *Annual International Cryptology Conference*. Springer, 2021, pp. 64–93.
- [12] I. Cascudo, B. David, L. Garms, and A. Konring, “Yolo yoso: fast and simple encryption and secret sharing in the yoso model,” in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2022, pp. 651–680.
- [13] R. Canetti, S. Kolby, D. Ravi, E. Soria-Vazquez, and S. Yakubov, “Taming adaptivity in yoso protocols: The modular way,” in *Theory of Cryptography Conference*. Springer, 2023, pp. 33–62.
- [14] L. Braun, I. Damgård, and C. Orlandi, “Secure multiparty computation from threshold encryption based on class groups,” in *Annual International Cryptology Conference*. Springer, 2023, pp. 613–645.
- [15] S. Kolby, D. Ravi, and S. Yakubov, “Constant-round yoso mpc without setup,” *Cryptology ePrint Archive*, 2022.
- [16] B. David, G. Deligios, A. Goel, Y. Ishai, A. Konring, E. Kushilevitz, C.-D. Liu-Zhang, and V. Narayanan, “Perfect mpc over layered graphs,” in *Annual International Cryptology Conference*. Springer, 2023, pp. 360–392.
- [17] A. Bandarupalli, X. Ji, A. Kate, C.-D. Liu-Zhang, and Y. Song, “Computationally efficient asynchronous mpc with linear communication and low additive overhead,” in *Annual International Cryptology Conference*. Springer, 2025, pp. 261–294.
- [18] A. Choudhuri, M. Hirt, and A. Patra, “Asynchronous multiparty computation with linear communication complexity,” in *International Symposium on Distributed Computing*. Springer, 2013, pp. 388–402.
- [19] A. Choudhuri and A. Patra, “An efficient framework for unconditionally secure multiparty computation,” *IEEE Transactions on Information Theory*, vol. 63, no. 1, pp. 428–468, 2016.
- [20] —, “On the communication efficiency of statistically secure asynchronous mpc with optimal resilience,” *Journal of Cryptology*, vol. 36, no. 2, p. 13, 2023.
- [21] V. Goyal, C.-D. Liu-Zhang, and Y. Song, “Towards achieving asynchronous mpc with linear communication and optimal resilience,” *Cryptology ePrint Archive*, 2024.
- [22] J. Groth and V. Shoup, “Design and analysis of a distributed ecads signing service,” *Cryptology ePrint Archive*, 2022.
- [23] Y. Su, Y. Lu, J. Li, Y. Wang, C. Dong, and Q. Tang, “Dumbo-mpc: efficient fully asynchronous mpc with optimal resilience,” in *USENIX Security*, vol. 2024, 2025.
- [24] S. Das, Z. Xiang, and L. Ren, “Asynchronous data dissemination and its applications,” in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2705–2721.
- [25] T. Yurek, L. Luo, J. Fairuze, A. Kate, and A. Miller, “hbaccs: How to robustly share many secrets,” in *Annual Network and Distributed System Security Symposium (NDSS)*. The Internet Society, 2022, pp. 24–28.
- [26] I. Abraham, G. Asharov, A. Patra, and G. Stern, “Perfectly secure asynchronous agreement on a core set in constant expected time,” *IACR Cryptol. ePrint Arch.*, vol. 2023, p. 1130, 2023.
- [27] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, “Secure and efficient asynchronous broadcast protocols,” in *Annual International Cryptology Conference*. Springer, 2001, pp. 524–541.
- [28] A. Kate, G. M. Zaverucha, and I. Goldberg, “Constant-size commitments to polynomials and their applications,” in *International conference on the theory and application of cryptography and information security*. Springer, 2010, pp. 177–194.
- [29] A. Bienstock, D. Escudero, and A. Polychroniadou, “Perfectly secure fluid mpc with abort and linear communication complexity,” *Cryptology ePrint Archive*, 2024.
- [30] G. Deligios, A. Konring, C.-D. Liu-Zhang, and V. Narayanan, “Statistical layered mpc,” in *Theory of Cryptography Conference*. Springer, 2025, pp. 362–394.
- [31] A. Erwig, S. Faust, and S. Riahi, “Large-scale non-interactive threshold cryptosystems in the yoso model,” *Cryptology ePrint Archive*, 2021.
- [32] C.-D. Liu-Zhang, E. Masserova, J. Ribeiro, P. Soni, and S. A. Thyagarajan, “Improved yoso randomness generation with worst-case corruptions,” *Cryptology ePrint Archive*, 2024.
- [33] A. Acharya, C. Hazay, V. Kolesnikov, and M. Prabhakaran, “Scales: Mpc with small clients and larger ephemeral servers,” in *Theory of Cryptography Conference*. Springer, 2022, pp. 502–531.
- [34] I. Damgård, D. Escudero, and A. Polychroniadou, “Phoenix: Secure computation in an unstable network with dropouts and comebacks,” *Cryptology ePrint Archive*, 2021.

- [35] R. Bhadauria, J. H.-y. Chiang, D. Ravi, J. Sternad, and S. Yakubov, “Minimizing the use of the honest majority in yoso mpc with guaranteed output delivery,” *Cryptology ePrint Archive*, 2024.
- [36] I. B. Damgård, S. H. Kamp, J. Loss, and J. B. Nielsen, “Asynchronous yoso a la paillier,” *Cryptology ePrint Archive*, 2025.
- [37] S. Das, Z. Xiang, and L. Ren, “Powers of tau in asynchrony,” in *Proceedings of the 2024 Network and Distributed System Security Symposium*, 2024.
- [38] M. Ben-Or, R. Canetti, and O. Goldreich, “Asynchronous secure computation,” in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 52–61.
- [39] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, “Dumbo: Faster asynchronous bft protocols,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 803–818.
- [40] I. Abraham, D. Malkhi, and A. Spiegelman, “Asymptotically optimal validated asynchronous byzantine agreement,” in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 337–346.
- [41] S. Duan, X. Wang, and H. Zhang, “Fin: practical signature-free asynchronous common subset in constant time,” in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 815–829.
- [42] D. Boneh, J. Drake, B. Fisch, and A. Gabizon, “Efficient polynomial commitment schemes for multiple points and polynomials,” *Cryptology ePrint Archive*, 2020.
- [43] J. Camenisch and M. Stadler, “Efficient group signature schemes for large groups,” in *Advances in Cryptology – CRYPTO ’97*, ser. Lecture Notes in Computer Science, vol. 1294. Springer, 1997, pp. 410–424.
- [44] —, “Proof systems for general statements about discrete logarithms,” *D-INFK Technical Report*, vol. 260, 1997.
- [45] A. Fiat and A. Shamir, “How to prove yourself: Practical solutions to identification and signature problems,” in *Conference on the theory and application of cryptographic techniques*. Springer, 1986, pp. 186–194.
- [46] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, “Bulletproofs: Short proofs for confidential transactions and more,” in *2018 IEEE symposium on security and privacy (SP)*. IEEE, 2018, pp. 315–334.
- [47] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Koffi, M. J. Fischer, and B. Ford, “Scalable bias-resistant distributed randomness,” in *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee, 2017, pp. 444–460.

APPENDIX A

SECURITY PROOFS FOR KZG POLYNOMIAL COMMITMENT WITH AGGREGATION INTERFACES

In this section, we formally prove that our proposed KZG scheme with aggregation interfaces satisfies the properties of *correctness*, *public verifiability*, *evaluation binding*, *polynomial binding*, and *hiding* under the hardness assumptions of the discrete logarithm and t -strong Diffie–Hellman problems, together with the completeness, zero-knowledge, and knowledge soundness of the underlying pok_{Ped} in the random-oracle model. We first formally define these properties and then prove that our construction satisfies them.

Correctness: Given any $SP \leftarrow \text{Setup}(1^\kappa, t)$ and any collection of polynomials $\{f_j(\cdot)\}_{j \in [B]} \subseteq \mathbb{Z}_q[x]$, let $(C_j, \hat{f}_j(\cdot)) \leftarrow \text{PolyCom}(SP, f_j(\cdot))$ be honestly generated for each $j \in [B]$, and let $\mathbf{C} = (C_1, \dots, C_B)$. For any evaluation point i and public challenge context ctx derived from the public transcript, define the Pedersen-style evaluation-commitment vector $\text{Ped} = (g^{f_1(i)} h^{\hat{f}_1(i)}, \dots, g^{f_B(i)} h^{\hat{f}_B(i)})$. Then:

- For each $j \in [B]$, if $w_{j,i} \leftarrow \text{ProveEval}(SP, i, f_j(\cdot), \hat{f}_j(\cdot))$, then $\text{VerifyEval}(SP, C_j, i, f_j(i), \hat{f}_j(i), w_{j,i}) = 1$.

- If $(T_i, W_i, \pi_i^{\text{ped}}) \leftarrow \text{AggPubProEval}(SP, i, \text{ctx}, \{f_j(\cdot), \hat{f}_j(\cdot)\}_{j \in [B]})$, then $\text{AggPubVerEval}(SP, i, \text{ctx}, T_i, W_i, \pi_i^{\text{ped}}, \mathbf{C}) = 1$.
- For the same W_i and the vector Ped defined above, $\text{AggPedVerEval}(SP, i, \text{ctx}, W_i, \text{Ped}, \mathbf{C}) = 1$.

Public verifiability: Consider any $SP \leftarrow \text{Setup}(1^\kappa, t)$ and any collection of polynomials $\{f_j(\cdot)\}_{j \in [B]} \subseteq \mathbb{Z}_q[x]$, with corresponding KZG commitment vector $\mathbf{C} = (C_1, \dots, C_B)$, where $(C_j, \hat{f}_j(\cdot)) \leftarrow \text{PolyCom}(SP, f_j(\cdot))$ for each $j \in [B]$. For any evaluation point i and public challenge context ctx derived from the public transcript, an external verifier can execute $\text{AggPubVerEval}(SP, i, \text{ctx}, T_i, W_i, \pi_i^{\text{ped}}, \mathbf{C})$ using only the public inputs $(SP, i, \text{ctx}, T_i, W_i, \pi_i^{\text{ped}}, \mathbf{C})$. If the Pedersen-style evaluation-commitment vector $\text{Ped} = (g^{f_1(i)} h^{\hat{f}_1(i)}, \dots, g^{f_B(i)} h^{\hat{f}_B(i)})$ is also publicly available, the verifier can likewise execute $\text{AggPedVerEval}(SP, i, \text{ctx}, W_i, \text{Ped}, \mathbf{C})$ using only public inputs. Neither verification interface requires access to the individual evaluations $\{f_j(i)\}_{j \in [B]}$ or their hiding randomness $\{\hat{f}_j(i)\}_{j \in [B]}$.

Evaluation binding: For any PPT adversary $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} SP \leftarrow \text{Setup}(1^\kappa, t); \\ (C, i, (f(i), \hat{f}(i), w_i), \\ (f'(i), \hat{f}'(i), w'_i)) \leftarrow \mathcal{A}(SP) : \\ \text{VerifyEval}(SP, C, i, f(i), \hat{f}(i), w_i) = 1 \wedge \\ \text{VerifyEval}(SP, C, i, f'(i), \hat{f}'(i), w'_i) = 1 \wedge \\ f(i) \neq f'(i) \end{array} \right] \leq \varepsilon(\kappa)$$

Polynomial binding: For any PPT adversary $\mathcal{A}$,

$$\Pr \left[\begin{array}{l} SP \leftarrow \text{Setup}(1^\kappa, t); \\ (I_1, I_2, C, \{f(i), \hat{f}(i), w_i\}_{i \in I_1 \cup I_2}) \leftarrow \mathcal{A}(SP) \\ \text{where } I_1, I_2 \subseteq [n], |I_1| = |I_2| = t + 1 : \\ \forall i \in I_1 \cup I_2, \\ \text{VerifyEval}(SP, C, i, f(i), \hat{f}(i), w_i) = 1 \wedge \\ f_1(\cdot) \leftarrow \sum_{i \in I_1} \lambda_i \cdot f(i) \wedge \\ \hat{f}_1(\cdot) \leftarrow \sum_{i \in I_1} \lambda_i \cdot \hat{f}(i) \wedge \\ f_2(\cdot) \leftarrow \sum_{i \in I_2} \lambda_i \cdot f(i) \wedge \\ \hat{f}_2(\cdot) \leftarrow \sum_{i \in I_2} \lambda_i \cdot \hat{f}(i) \wedge f_1(\cdot) \neq f_2(\cdot) \end{array} \right] \leq \varepsilon(\kappa)$$

Hiding: For each aggregated evaluation at point i , let tr_i denote the public transcript available before the corresponding aggregated evaluation proof is generated. Let Ctx denote the deterministic polynomial-time algorithm used to derive the public challenge context, and set $\text{ctx}_i = \text{Ctx}(SP, \text{tr}_i)$. The commitments together with their individual and aggregated evaluation proofs reveal no information about the underlying polynomials beyond the evaluations explicitly disclosed to the adversary and the corresponding public information. Formally, there exist PPT simulators ($\text{Sim}_0, \text{Sim}_1, \text{Sim}_2, \text{Sim}_3$) such that, for every PPT adversary $\mathcal{A}$, the following two distributions are computationally indistinguishable:

Real World:

$$\left\{ \begin{array}{l} SP \leftarrow \text{Setup}(1^\kappa, t); (\{f_j(\cdot)\}_{j \in [B]}, I_1, I_2) \leftarrow \mathcal{A}(SP); \\ \text{where } I_1, I_2 \subseteq [n], |I_1| \leq t, |I_2| \leq n; \\ \forall j \in [B], (C_j, \hat{f}_j(\cdot)) \leftarrow \text{PolyCom}(SP, f_j(\cdot)); \\ \mathbf{C} \leftarrow (C_1, \dots, C_B); \forall i \in I_1, \forall j \in [B], \\ w_i \leftarrow \text{ProveEval}(SP, i, f_j(\cdot), \hat{f}_j(\cdot)); \\ \forall i \in I_2, \text{ctx}_i \leftarrow \text{Ctx}(SP, \text{tr}_i), \\ (T_i, W_i, \pi_i^{\text{ped}}) \leftarrow \text{AggPubProEval}(SP, i, \text{ctx}_i, \\ \{f_j(\cdot), \hat{f}_j(\cdot)\}_{j \in [B]}) \\ : (\mathbf{C}, \{i, f_j(i), \hat{f}_j(i), w_{ji}\}_{i \in I_1, j \in [B]}, \\ \{i, \text{ctx}_i, T_i, W_i, \pi_i^{\text{ped}}\}_{i \in I_2}) \end{array} \right\}$$

Ideal World:

$$\left\{ \begin{array}{l} (SP, st) \leftarrow \text{Sim}_0(1^\kappa, t); \\ (\{f_j(\cdot)\}_{j \in [B]}, I_1, I_2) \leftarrow \mathcal{A}(SP); \\ \text{where } I_1, I_2 \subseteq [n], |I_1| \leq t, |I_2| \leq n; \\ \forall j \in [B], (C_j, c_j) \leftarrow \text{Sim}_1(st), \mathbf{C} \leftarrow (C_1, \dots, C_B); \\ \forall j \in [B], \\ (\{w_{j,i}\}_{i \in I_1}, f'_j(\cdot), \hat{f}'_j(\cdot)) \leftarrow \text{Sim}_2(st, C_j, c_j, \\ \{i, f_j(i), \hat{f}_j(i)\}_{i \in I_1}); \\ \forall i \in I_2, \text{ctx}_i \leftarrow \text{Ctx}(SP, \text{tr}_i), \\ (T_i, W_i, \pi_i^{\text{ped}}) \leftarrow \text{Sim}_3(SP, st, i, \text{ctx}_i, \\ \{f'_j(\cdot), \hat{f}'_j(\cdot)\}_{j \in [B]}) \\ : (\mathbf{C}, \{i, f_j(i), \hat{f}_j(i), w_{ji}\}_{i \in I_1, j \in [B]}, \\ \{i, \text{ctx}_i, T_i, W_i, \pi_i^{\text{ped}}\}_{i \in I_2}) \end{array} \right\}$$

We now state and prove the main security theorem.

Theorem 1. *Assuming the discrete logarithm assumption and the t -strong Diffie–Hellman assumption over the bilinear group, and assuming that pok_{Ped} is a non-interactive zero-knowledge proof of knowledge satisfying completeness, zero-knowledge, and knowledge soundness in the random-oracle model, the augmented KZG scheme with aggregation interfaces satisfies correctness, public verifiability, evaluation binding, polynomial binding, and hiding with overwhelming probability.*

Proof. We establish each property individually.

Correctness. For a single evaluation, correctness follows directly from the standard KZG evaluation verification equation. For an aggregated evaluation, the prover and verifier derive the same challenge $\gamma = \mathcal{H}(SP, \text{ctx})$. By the definitions of C , T_i , and W_i in Figure 5, they satisfy the aggregated KZG pairing equation. Moreover, by the completeness of pok_{Ped} , the honestly generated π_i^{ped} is accepted by AggPubVerEval . For AggPedVerEval , aggregating the Pedersen-style evaluation commitments using the same challenge γ yields the same T_i , and hence the same pairing equation is satisfied. Therefore, VerifyEval , AggPubVerEval , and AggPedVerEval all output 1 on honestly generated inputs.

Public Verifiability. All inputs to $\text{AggPubVerEval}(SP, i, \text{ctx}, T_i, W_i, \pi_i^{\text{ped}}, \mathbf{C})$ are publicly available. Hence, an external verifier can recompute the challenge γ , aggregate $\mathbf{C}$, verify π_i^{ped} , and check the aggregated KZG pairing equation

without access to the individual evaluations or hiding randomness. Likewise, when Ped is publicly available, all inputs to $\text{AggPedVerEval}(SP, i, \text{ctx}, W_i, \text{Ped}, \mathbf{C})$ are public, and the verifier can aggregate Ped and $\mathbf{C}$ using the same challenge and check the corresponding pairing equation. Therefore, both verification interfaces can be executed using only the public transcript.

Evaluation Binding. Suppose there exists a PPT adversary $\mathcal{A}$ that breaks evaluation binding, i.e., it outputs two distinct tuples $(i, f(i), \hat{f}(i), w_i)$ and $(i, f'(i), \hat{f}'(i), w'_i)$ for the same evaluation point i , both of which are accepted by VerifyEval with non-negligible probability. Then an algorithm $\mathcal{B}$ can use $\mathcal{A}$ as a subroutine to break the t -SDH assumption with non-negligible probability. The reduction follows directly from the original KZG proof [28], and we omit the details here.

Polynomial Binding. Suppose that there exists a PPT adversary $\mathcal{A}$ that outputs a commitment C together with two distinct degree- t polynomials that both verify under VerifyEval with non-negligible probability. As shown in the original KZG proof [28], this would either imply a violation of evaluation binding or allow the extraction of two different polynomials committed to the same C , which directly yields an algorithm for solving the t -SDH or discrete logarithm assumption. Therefore, polynomial binding holds under these hardness assumptions.

Hiding. We prove hiding by a sequence of games showing that the adversary's view is computationally indistinguishable between the real protocol and the ideal world. Let Sim_{pok} denote the PPT zero-knowledge simulator for pok_{Ped} in the random-oracle model.

- **Game 0:** This is the real-world execution.
- **Game 1:** Same as Game 0, except that the trapdoors α and τ are sampled uniformly from $\mathbb{Z}_q$ and retained by the simulator. Since the distribution of SP is identical to that in Game 0, the two games are identically distributed.
- **Game 2:** Same as Game 1, except that for every $i \in I_2$, the honestly generated proof π_i^{ped} is replaced by $\pi_i^{\text{ped}} \leftarrow \text{Sim}_{\text{pok}}(SP, \text{ctx}_i, T_i)$. By the zero-knowledge property of pok_{Ped} , jointly replacing the polynomially many proofs changes the adversary's view only negligibly. Therefore, Game 2 is computationally indistinguishable from Game 1.
- **Game 3:** Same as Game 2, except that the commitments and evaluation witnesses are generated by Sim_1 and Sim_2 , respectively, and for every $i \in I_2$, the aggregated transcript $(T_i, W_i, \pi_i^{\text{ped}})$ is generated by Sim_3 , as defined in Figure 12. Since Ctx is deterministic, applying it to the corresponding simulated public transcript preserves the distribution of the challenge contexts. Hence, Game 3 is identically distributed to Game 2.

Combining the above game transitions, the adversary's view in the real world is computationally indistinguishable from its view in the ideal world, and the hiding property follows. $\square$

Simulator for KZG with Aggregation

Let Sim_{pok} denote the PPT zero-knowledge simulator for pok_{Ped} in the random-oracle model.

- 1) $\text{Sim}_0(1^\kappa, t)$: choose random trapdoors (α, τ) , run $\text{Setup}(1^\kappa, t)$ with them, and output (SP, st) where $st = (\alpha, \tau)$.
- 2) $\text{Sim}_1(st)$: for each $j \in [B]$, sample $c_j \xleftarrow{\$} \mathbb{Z}_q$, set $C_j \leftarrow g^{c_j}$, and return (C_j, c_j) .
- 3) $\text{Sim}_2(st, C_j, c_j, \{i, f_j(i)\}_{i \in I_1})$:
 - a) set $Q \leftarrow I_1$ and $Q' \leftarrow \emptyset$. Sample $i \in [n] \setminus I_1$, $f_j(i) \xleftarrow{\$} \mathbb{Z}_q$ and set $Q \leftarrow Q \cup \{i\}$ until $Q > t$.
 - b) interpolate $f'_j(\cdot)$ from $\{i, f_j(i)\}_{i \in Q}$.
 - c) compute $\hat{f}'_j(\alpha) \leftarrow \frac{c_j - f'_j(\alpha)}{\tau}$.
 - d) set $Q' \leftarrow \{\alpha\}$. Sample $k \in [n]$, $\hat{f}'_j(k) \xleftarrow{\$} \mathbb{Z}_q$ and set $Q' \leftarrow Q' \cup \{k\}$ until $Q' > t$.
 - e) interpolate $\hat{f}'_j(\cdot)$ from $\{k, \hat{f}'_j(k)\}_{k \in Q'}$.
 - f) for $\forall i \in I_1$, compute $w_{j,i} \leftarrow g^{\frac{f'_j(\alpha) - f'_j(i)}{\alpha - i}} h^{\frac{\hat{f}'_j(\alpha) - \hat{f}'_j(i)}{\alpha - i}}$.
 - g) return $(\{w_{j,i}\}_{i \in I_1}, f'_j(\cdot), \hat{f}'_j(\cdot))$.
- 4) $\text{Sim}_3(SP, st, i, \text{ctx}, \{f'_j(\cdot), \hat{f}'_j(\cdot)\}_{j \in [B]})$:
 - a) compute $\gamma \leftarrow \mathcal{H}(SP, \text{ctx}_i)$.
 - b) compute $T_i \leftarrow \prod_{j \in [B]} (g^{\frac{f'_j(\alpha) - f'_j(i)}{\alpha - i}} h^{\frac{\hat{f}'_j(\alpha) - \hat{f}'_j(i)}{\alpha - i}})^{\gamma^{j-1}}$, and $W_i \leftarrow \prod_{j \in [B]} (g^{\frac{f'_j(\alpha) - f'_j(i)}{\alpha - i}} h^{\frac{\hat{f}'_j(\alpha) - \hat{f}'_j(i)}{\alpha - i}})^{\gamma^{j-1}}$.
 - c) generate $\pi_i^{\text{ped}} \leftarrow \text{Sim}_{\text{pok}}(SP, \text{ctx}_i, T_i)$.
 - d) return $(\{T_i, W_i, \pi_i^{\text{ped}}\}_{i \in I_2})$.

Fig. 12. Simulator ($\text{Sim}_0, \text{Sim}_1, \text{Sim}_2, \text{Sim}_3$) for KZG polynomial commitment with aggregation.

APPENDIX B

SECURITY ANALYSIS OF AggTrans

Theorem 2 (Termination, Transfer Consistency, and Secrecy of AggTrans). *If all honest servers in $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$ execute the AggTrans, then, with all but negligible probability, the following properties hold:*

- **Termination:** All honest servers in $\mathcal{P}^{l+1}$ eventually obtain new_shares together with the corresponding randomness shares, evaluation proofs, and commitments.
- **Transfer Consistency:** For every gate computed in epoch C^l , if committee $\mathcal{P}^l$ holds a degree- t sharing $[s]^l$ encoding the secret s , then all honest servers in the next committee $\mathcal{P}^{l+1}$ reconstruct a fresh degree- t sharing $[s]^{l+1}$ encoding the same secret s .
- **Secrecy:** For the output new_shares, the adversary corrupting up to t servers in each of $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$ learns no additional information about $\{s_k\}_{k \in [B]}$ beyond what is revealed by the corrupted servers' own shares.

Lemma 1 (Termination). *All honest servers in $\mathcal{P}^{l+1}$ eventually obtain new_shares together with the corresponding randomness shares, evaluation proofs, and commitments.*

Proof. The AggTrans protocol involves two main communication components: BACSS and MVBA. During the hand-off, the R-adaptive adversary $\mathcal{A}$ may adaptively corrupt at most $t < n/3$ servers in each of $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$; hence, $\mathcal{P}^l$ contains

at least $n - t \geq 2t + 1$ honest dealers. By the correctness and completeness of BACSS and the eventual delivery of messages between honest parties, every honest server in $\mathcal{P}^{l+1}$ eventually receives valid shares from these honest dealers. Since the honest dealers use the same prior-epoch commitment vector, every honest server in $\mathcal{P}^{l+1}$ eventually obtains an externally valid dealer set of size $2t + 1$ and invokes MVBA with a valid input. Moreover, $\mathcal{P}^{l+1}$ contains at least $n - t \geq 2t + 1$ honest servers. By the termination property of MVBA, all honest servers in $\mathcal{P}^{l+1}$ will eventually agree on the same common subset $T \subseteq \mathcal{P}^l$ of size $2t + 1$. Once this subset T is fixed, each honest server P_j^{l+1} can locally interpolate its new shares $\{[s_k]_j^{l+1}, [\hat{s}_k]_j^{l+1}\}_{k \in [B]}$, and derive the corresponding evaluation proofs $\{w_{kj}^{l+1}\}_{k \in [B]}$ and commitments $\{C_k^{l+1}\}_{k \in [B]}$. Hence, all honest servers in $\mathcal{P}^{l+1}$ complete the protocol, thereby terminating AggTrans. $\square$

Lemma 2 (Aggregated Anchoring Soundness). *For any dealer P_i^l , suppose that an honest server accepts both AggPubVerEval checks for P_i^l in Algorithm 1, where the new commitment vector $\mathbf{C}_i^{l+1}$ is verified at point 0 and the prior-epoch commitment vector $\mathbf{C}^l$ is verified at point i using the same aggregated evaluation commitment $T_{i,0}^{l+1}$ and common challenge context ctx_i . Then, with all but negligible probability, $f_{ki}^{l+1}(0) = f_k^l(i)$ for every $k \in [B]$.*

Proof. Fix a dealer P_i^l whose two AggPubVerEval checks are accepted. Since ctx_i contains the ordered commitment vectors $\mathbf{C}_i^{l+1}$ and $\mathbf{C}^l$, polynomial binding ensures that their committed polynomials, and hence their evaluations, are fixed before $\gamma_i = \mathcal{H}(SP, \text{ctx}_i)$ is returned. By the knowledge soundness of pok_{Ped} , the accepted proof yields an opening $(v_i, \hat{v}_i)$ of the common $T_{i,0}^{l+1}$. The homomorphic property and evaluation binding then imply

$$v_i = \sum_{k \in [B]} \gamma_i^{k-1} f_{ki}^{l+1}(0) = \sum_{k \in [B]} \gamma_i^{k-1} f_k^l(i).$$

Let $D_i(X) = \sum_{k \in [B]} (f_{ki}^{l+1}(0) - f_k^l(i)) X^{k-1}$. If any corresponding evaluations differ, then $D_i(X)$ is non-zero and has degree at most $B - 1$, while acceptance requires $D_i(\gamma_i) = 0$. Conditioned on the adversary's view before each random-oracle query, $D_i(X)$ is fixed and γ_i is uniform and independent. Thus, the success probability is at most $(B - 1)/q$ per query and remains negligible after a union bound over polynomially many adaptive queries. Consequently, with all but negligible probability, $f_{ki}^{l+1}(0) = f_k^l(i)$ for every $k \in [B]$. $\square$

Lemma 3 (Transfer Consistency). *For every gate computed in epoch C^l , if committee $\mathcal{P}^l$ holds a degree- t sharing $[s]^l$ encoding the secret s , then all honest servers in the next committee $\mathcal{P}^{l+1}$ reconstruct a fresh degree- t sharing $[s]^{l+1}$ encoding the same secret s .*

Proof. By the completeness of BACSS, once an honest server accepts the BACSS instance of a dealer P_i^l , all honest servers eventually receive shares consistent with the same degree- t resharing polynomial $f_{ki}^{l+1}(\cdot)$ for every $k \in [B]$. In particular,

each honest server P_j^{l+1} receives $[[s_k]_i]_j^{l+1} = f_{ki}^{l+1}(j)$, which is verified against the corresponding commitment C_{ki}^{l+1} . Moreover, since an accepted dealer passes both AggPubVerEval checks in Algorithm 1, Lemma 2 guarantees that, except with negligible probability, $f_{ki}^{l+1}(0) = f_k^l(i) = [s_k]_i^l$. Thus, each resharing polynomial encodes the corresponding prior-epoch share as its constant term.

By the agreement and external-validity properties of MVBA, all honest servers agree on the same subset T of accepted dealers satisfying the matching-commitment condition. Thus, for every $i \in T$, the value $f_{ki}^{l+1}(0)$ is anchored to the corresponding evaluation $f_k^l(i)$ of the same degree- t polynomial committed by C^l . Lagrange interpolation over T therefore yields a fresh degree- t sharing of $f_k^l(0) = s_k$ for every $k \in [B]$. $\square$

Lemma 4 (Secrecy). *For the output `new_shares`, the adversary corrupting up to t servers in each of $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$ learns no additional information about $\{s_k\}_{k \in [B]}$ beyond what is revealed by the corrupted servers' own shares.*

Proof. We prove secrecy by constructing a sequence of game-based simulations, showing that a real-world adversary cannot distinguish the actual protocol execution of AggTrans from a simulated one. Let $\mathcal{H}^l$ (resp. $\mathcal{H}^{l+1}$) be the honest servers in $\mathcal{P}^l$ (resp. $\mathcal{P}^{l+1}$), and let $\mathcal{A}^l$ (resp. $\mathcal{A}^{l+1}$) be the corrupted servers in each committee.

Game 0. The adversary corrupts up to t servers in $\mathcal{P}^l$ (forming $\mathcal{A}^l$) and up to t servers in $\mathcal{P}^{l+1}$ (forming $\mathcal{A}^{l+1}$). The honest servers $\mathcal{H}^l \cup \mathcal{H}^{l+1}$ execute the protocol exactly as prescribed by AggTrans . This represents the real-world protocol execution.

Game 1. The simulator first samples the common reference string (CRS) by choosing $\alpha, \tau \xleftarrow{\$} \mathbb{Z}_q$ and publishing $\{g, h = g^\tau, (g^\alpha, \dots, g^{\alpha^t}), (h^\alpha, \dots, h^{\alpha^t})\}$. It then impersonates all honest servers in $\mathcal{H}^l \cup \mathcal{H}^{l+1}$. For each honest dealer $P_i^l \in \mathcal{H}^l$, the simulator picks a random degree- t polynomial $f_i(\cdot)$ in $\mathbb{Z}_q[x]$ with $f_i(0) = [s]_i^l$. It uses $f_i(\cdot)$ as input to BACSS_share_i and invokes AggPubProEval in exactly the same way as honest servers in $\mathcal{H}^l$. The simulator also runs MVBA faithfully, ensuring that any corrupted server $P_j^{l+1} \in \mathcal{A}^{l+1}$ receives a set of shares $\{[[s]_i]_j^{l+1}\}_{i \in T}$ consistent with an agreed-upon subset $T \subseteq \mathcal{P}^l$, where $[[s]_i]_j^{l+1} = f_i(j)$.

Since the simulator follows the protocol specification exactly (from the viewpoint of any uncorrupted server), the adversary's view in **Game 1** is computationally indistinguishable from that in **Game 0**.

Game 2. Here, the simulator modifies how it provides shares to corrupted servers. Rather than genuinely invoking BACSS_share_i for each honest dealer P_i^l , the simulator takes as input the corrupted shares $\{[[s]_i]_j^{l+1}\}_{i \in \mathcal{H}^l, j \in \mathcal{A}^{l+1}}$ that the adversary would see. It then randomly samples a degree- t polynomial $f'(\cdot)$. For each $P_i^l \in \mathcal{H}^l$, the simulator interpolates a polynomial $f'_i(\cdot)$ consistent with these corrupted shares (so that $f'_i(j) = [[s]_i]_j^{l+1}$) and sets $f'_i(0) = f'(i)$ to fix any remaining degrees of freedom.

From the adversary's perspective, this change does not alter the distribution of the shares it observes. Moreover, the hiding property of the augmented KZG commitments ensures that, beyond the shares explicitly given to corrupted servers, no additional information about the underlying polynomials is revealed. Thus, the view of corrupted servers in **Game 2** remains indistinguishable from **Game 1**. Since **Game 1** is also indistinguishable from **Game 0**, it follows that AggTrans preserves secrecy against an adversary corrupting up to t parties in each of $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$. $\square$

APPENDIX C

HADAMARD-PRODUCT PROOF FOR IPA_{KZG}

We detail how IPA_{KZG} in Figure 7 proves the Hadamard-product relation over the committed polynomial evaluations. We first formalize the committed Hadamard-product relation, and then show how it is reduced to the arithmetic-circuit relation of Bulletproofs [46]. Finally, we explain how the same committed values are linked to their corresponding KZG polynomial evaluations.

Committed Hadamard-product relation. For each $x \in \{a, b, c\}$ and $k \in [M]$, let $x_k = f_{x_k}(i_x)$ and $\hat{x}_k = f_{x_k}(i_x)$ denote the corresponding polynomial evaluation and hiding value, respectively. Define the Pedersen-style evaluation commitment $\text{Ped}_{x,k} = g^{x_k} h^{\hat{x}_k}$, and let $\text{Ped}_x = (\text{Ped}_{x,1}, \dots, \text{Ped}_{x,M})$ denote the corresponding commitment vector. The committed Hadamard-product relation is

$$\left\{ \begin{array}{l} \text{public : } SP, \{(\text{Ped}_{a,k}, \text{Ped}_{b,k}, \text{Ped}_{c,k}) \in \mathbb{G}^3\}_{k \in [M]}; \\ \text{witness : } \{(a_k, \hat{a}_k, b_k, \hat{b}_k, c_k, \hat{c}_k) \in \mathbb{Z}_q^6\}_{k \in [M]} \\ \text{relation : } \forall k \in [M], c_k = a_k \cdot b_k \wedge \text{Ped}_{a,k} = g^{a_k} h^{\hat{a}_k} \\ \quad \wedge \text{Ped}_{b,k} = g^{b_k} h^{\hat{b}_k} \wedge \text{Ped}_{c,k} = g^{c_k} h^{\hat{c}_k}. \end{array} \right\}$$

Equivalently, letting $\mathbf{a} = (a_1, \dots, a_M)^\top$, $\mathbf{b} = (b_1, \dots, b_M)^\top$, and $\mathbf{c} = (c_1, \dots, c_M)^\top$, the multiplicative constraint is the Hadamard-product relation $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$.

Bulletproof arithmetic-circuit variant. We use the arithmetic-circuit variant of Bulletproofs (Section 5 of [46]), which provides a proof system for the following relation over committed values:

$$\left\{ \begin{array}{l} \text{public : } SP, \text{Ped} \in \mathbb{G}^V, \mathbf{A}_a, \mathbf{A}_b, \mathbf{A}_c \in \mathbb{Z}_q^{Q \times M}, \\ \quad \mathbf{A}_v \in \mathbb{Z}_q^{Q \times V}, \chi \in \mathbb{Z}_q^Q; \\ \text{witness : } \mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{Z}_q^M, \mathbf{v}, \hat{\mathbf{v}} \in \mathbb{Z}_q^V; \\ \text{relation : } \text{Ped}_r = g^{v_r} h^{\hat{v}_r}, \quad \forall r \in [V] \wedge \mathbf{a} \circ \mathbf{b} = \mathbf{c} \\ \quad \wedge \mathbf{A}_a \mathbf{a} + \mathbf{A}_b \mathbf{b} + \mathbf{A}_c \mathbf{c} = \mathbf{A}_v \mathbf{v} + \chi. \end{array} \right\}$$

Here, M denotes the number of multiplication gates, V denotes the number of externally committed values, and Q denotes the number of linear consistency constraints. The matrices $\mathbf{A}_a$, $\mathbf{A}_b$, and $\mathbf{A}_c$ specify the coefficients of the left-input, right-input, and output wires, respectively, while $\mathbf{A}_v$ specifies the coefficients of the values committed in Ped and χ is a public constant vector.

Reduction to the Bulletproof relation. For the committed Hadamard-product relation above, each multiplication gate contributes three externally committed values. We therefore set $V = 3M$ and $Q = 3M$, and arrange the committed values and their hiding randomness in block order as $\mathbf{v} = (\mathbf{a} \parallel \mathbf{b} \parallel \mathbf{c})$, $\hat{\mathbf{v}} = (\hat{\mathbf{a}} \parallel \hat{\mathbf{b}} \parallel \hat{\mathbf{c}})$ with $\mathbf{Ped} = (\mathbf{Ped}_a \parallel \mathbf{Ped}_b \parallel \mathbf{Ped}_c)$.

Let I_M and 0_M denote the $M \times M$ identity and all-zero matrices, respectively. We instantiate the linear consistency matrices as

$$\mathbf{A}_a = \begin{pmatrix} I_M \\ 0_M \\ 0_M \end{pmatrix}, \quad \mathbf{A}_b = \begin{pmatrix} 0_M \\ I_M \\ 0_M \end{pmatrix}, \quad \mathbf{A}_c = \begin{pmatrix} 0_M \\ 0_M \\ I_M \end{pmatrix},$$

and set $\mathbf{A}_v = I_{3M}$, $\chi = \mathbf{0}_{3M}$. Under this instantiation, the linear consistency constraint becomes

$$\begin{pmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{c} \end{pmatrix} = \mathbf{v},$$

thereby binding the multiplication-gate wires to their corresponding Pedersen-style commitments. Consequently, the remaining multiplicative constraint is exactly the committed Hadamard-product relation $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$.

Linking the Bulletproof relation to KZG-committed evaluations. The reduction above ensures that the openings of $\mathbf{Ped}_a$, $\mathbf{Ped}_b$, and $\mathbf{Ped}_c$ satisfy the committed Hadamard-product relation $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$. However, the Bulletproof argument alone only constrains the openings of these Pedersen-style commitments; it does not ensure that the committed values are the evaluations at i_x of the polynomials committed by $\mathbf{C}_x = (C_{x_1}, \dots, C_{x_M})$ for each $x \in \{a, b, c\}$.

To establish this link, IPA_{KZG} uses the same Pedersen-style commitment vector $\mathbf{Ped}_x$ in both the Bulletproof statement and the corresponding AggPedVerEval check. Specifically, let

$$\text{ctx} = (\text{IPAKZG}, i_a, i_b, i_c, \mathbf{C}_a, \mathbf{C}_b, \mathbf{C}_c, \mathbf{Ped}_a, \mathbf{Ped}_b, \mathbf{Ped}_c).$$

The prover derives $\gamma = \mathcal{H}(SP, \text{ctx})$ and, for each $x \in \{a, b, c\}$, aggregates the corresponding KZG evaluation witnesses as $W_x = \prod_{k \in [M]} w_{x_k}^{\gamma^{k-1}}$. The verifier then checks

$$\text{AggPedVerEval}(SP, i_x, \text{ctx}, W_x, \mathbf{Ped}_x, \mathbf{C}_x) = 1$$

for each $x \in \{a, b, c\}$.

Since ctx contains both the ordered KZG commitment vectors and the Pedersen-style commitment vectors, these vectors are fixed before the challenge γ is determined. The accepted AggPedVerEval checks therefore establish random-linear-combination consistency between the values constrained by the Bulletproof argument and the corresponding KZG polynomial evaluations. The following lemma formalizes that this aggregate consistency implies the required component-wise product relation.

Lemma 5 (Product-Relation Soundness). *For any proof accepted by IPA_{KZG} with respect to the KZG commitment vectors $\mathbf{C}_a, \mathbf{C}_b, \mathbf{C}_c$ and evaluation points i_a, i_b, i_c , let $\mathbf{x} = (f_{x_k}(i_x))_{k \in [M]}$ for each $x \in \{a, b, c\}$. Then, except with negligible probability, $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$.*

Proof. Fix an accepted proof. By the knowledge soundness of the Bulletproof arithmetic-circuit argument [46], there exists an efficient extractor that, except with negligible probability, extracts value vectors $\mathbf{a}', \mathbf{b}', \mathbf{c}'$ and the corresponding randomness opening the Pedersen-style commitment vectors. Moreover, the extracted values satisfy $a'_k \cdot b'_k = c'_k$ for every $k \in [M]$. Fix $x \in \{a, b, c\}$ and define

$$D_x(X) = \sum_{k \in [M]} (x'_k - f_{x_k}(i_x)) X^{k-1}.$$

The context ctx contains the ordered KZG and Pedersen-style commitment vectors. By polynomial binding of the KZG commitments and computational binding of the Pedersen-style commitments, both the committed polynomial evaluations and the extracted openings are fixed before $\gamma = \mathcal{H}(SP, \text{ctx})$ is determined. Acceptance of the corresponding AggPedVerEval check implies

$$\sum_{k \in [M]} \gamma^{k-1} x'_k = \sum_{k \in [M]} \gamma^{k-1} f_{x_k}(i_x),$$

and therefore $D_x(\gamma) = 0$. If the two evaluation vectors differ, then $D_x(X)$ is a non-zero polynomial of degree at most $M-1$. Conditioned on the adversary's view before each random-oracle query, $D_x(X)$ is fixed and γ is uniformly random and independent. Thus, the probability that $D_x(\gamma) = 0$ is at most $(M-1)/q$ per query. A union bound over $x \in \{a, b, c\}$ and polynomially many adaptive queries makes this probability negligible. Consequently, $x'_k = f_{x_k}(i_x)$ for every $x \in \{a, b, c\}$ and $k \in [M]$. Together with $a'_k \cdot b'_k = c'_k$, this gives $\mathbf{a} \circ \mathbf{b} = \mathbf{c}$. $\square$

APPENDIX D SECURITY ANALYSIS OF BatchMul

Theorem 3 (Termination, Multiplication Correctness, and Secrecy of BatchMul). *If all honest servers in $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$ execute the BatchMul, then, with all but negligible probability, the following properties hold:*

- *Termination: All honest servers in $\mathcal{P}^{l+1}$ eventually obtain `mul_shares` together with the corresponding randomness shares, evaluation proofs, and commitments.*
- *Multiplication Correctness: For each multiplication gate with inputs a and b , shared among $\mathcal{P}^l$ as $[a]^l$ and $[b]^l$, let $\langle c \rangle^l$ be the degree- $2t$ sharing of their product $c = a \cdot b$. Whenever $\mathcal{P}^l$ reshapes $\langle c \rangle^l$, all honest servers in $\mathcal{P}^{l+1}$ reconstruct a new degree- t sharing $[c]^{l+1}$ whose constant term equals the correct product c .*
- *Secrecy: For the output `mul_shares` = $\{[c_k]^{l+1}\}_{k \in [M]}$, an adversary corrupting up to t servers in each of $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$ learns no information about the secrets $\{c_k\}_{k \in [M]}$ beyond what can be inferred from the shares held by the corrupted servers.*

Lemma 6 (Termination). *All honest servers in $\mathcal{P}^{l+1}$ eventually obtain `mul_shares` together with the corresponding randomness shares, evaluation proofs, and commitments.*

Proof. In the BatchMul protocol, all n parallel BACSS instances are activated in parallel. During the hand-off, the R-adaptive adversary $\mathcal{A}$ may adaptively corrupt at most $t < n/3$ servers in each of $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$; hence, $\mathcal{P}^l$ contains at least $n - t \geq 2t + 1$ honest dealers. By the correctness and completeness of BACSS and the eventual delivery of messages between honest parties, every honest server in $\mathcal{P}^{l+1}$ eventually receives valid shares from these honest dealers. Since the honest dealers generate valid IPA_{KZG} proofs with respect to the same input commitment vectors, every honest server in $\mathcal{P}^{l+1}$ eventually obtains an externally valid dealer set of size $2t + 1$ and invokes MVBA with a valid input. Moreover, $\mathcal{P}^{l+1}$ contains at least $n - t \geq 2t + 1$ honest servers. By the termination of MVBA, all honest servers in $\mathcal{P}^{l+1}$ eventually agree on the same common subset of dealers. Based on this subset, they interpolate to obtain the new mul_shares together with the corresponding randomness shares, evaluation proofs, and commitments, thereby completing the protocol. $\square$

Lemma 7 (Multiplication Correctness). *For each multiplication gate with inputs a and b , shared among $\mathcal{P}^l$ as $[a]^l$ and $[b]^l$, let $\langle c \rangle^l$ be the degree- $2t$ sharing of their product $c = a \cdot b$. Whenever $\mathcal{P}^l$ reshapes $\langle c \rangle^l$, all honest servers in $\mathcal{P}^{l+1}$ reconstruct a new degree- t sharing $[c]^{l+1}$ whose constant term equals the correct product c .*

Proof. By the completeness of BACSS, for each accepted dealer P_i^l and each $k \in [M]$, all honest servers receive evaluations of the same degree- t resharing polynomial $f_{c_k}^{l+1}(\cdot)$, verified against $C_{c_k}^{l+1}$. The corresponding IPA_{KZG} instance uses $i_a = i_b = i$ and $i_c = 0$. Therefore, by Lemma 5, except with negligible probability,

$$f_{c_k}^{l+1}(0) = \langle c_k \rangle_i^l = [a_k]_i^l \cdot [b_k]_i^l$$

for every $k \in [M]$.

By the agreement and external-validity properties of MVBA, all honest servers agree on the same subset $T \subseteq \mathcal{P}^l$ of size $2t + 1$, and every dealer in T is verified with respect to the same input commitment vectors C_a^l and C_b^l . By polynomial binding, the accepted values $\{[a_k]_i^l, [b_k]_i^l\}_{i \in T}$ are evaluations of the same degree- t input sharings. Their products $\{\langle c_k \rangle_i^l\}_{i \in T}$ therefore lie on the corresponding degree- $2t$ product sharing. Since T contains $2t + 1$ evaluations of the degree- $2t$ product sharing, the honest servers can use the Lagrange coefficients determined by T to combine the corresponding degree- t resharnings, obtaining a degree- t sharing $[c_k]^{l+1}$ whose constant term is $a_k \cdot b_k$. Thus, BatchMul satisfies multiplication correctness. $\square$

Lemma 8 (Secrecy). *For the output $\text{mul\_shares} = \{[c_k]^{l+1}\}_{k \in [M]}$, an adversary corrupting up to t servers in each of $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$ learns no information about the secrets $\{c_k\}_{k \in [M]}$ beyond what can be inferred from the shares held by the corrupted servers.*

Proof. We establish secrecy via a sequence of games, showing that a real-world adversary cannot distinguish between the

actual protocol execution and a simulated one. Let $\mathcal{H}^l$ and $\mathcal{H}^{l+1}$ be the sets of honest servers in committees $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$, respectively. Let $\mathcal{A}^l$ and $\mathcal{A}^{l+1}$ be the corresponding sets of corrupted servers.

Game 0. The adversary corrupts up to t servers in $\mathcal{P}^l$ (forming $\mathcal{A}^l$) and up to t servers in $\mathcal{P}^{l+1}$ (forming $\mathcal{A}^{l+1}$). The honest servers $\mathcal{H}^l \cup \mathcal{H}^{l+1}$ execute the protocol exactly as prescribed by BatchMul. This represents the real-world protocol execution.

Game 1. The simulator first samples the common reference string (CRS) by choosing $\alpha, \tau \xleftarrow{\$} \mathbb{Z}_q$ and publishing $\{g, h = g^\tau, (g^\alpha, \dots, g^{\alpha^t}), (h^\alpha, \dots, h^{\alpha^t})\}$. It then impersonates all honest servers in $\mathcal{H}^l \cup \mathcal{H}^{l+1}$. For each honest dealer $P_i^l \in \mathcal{H}^l$, the simulator samples a random degree- t polynomial $f_{ci}(\cdot)$ in $\mathbb{Z}_q$ with $f_{ci}(0) = \langle c \rangle_i^l$. It uses $f_{ci}(\cdot)$ as input to BACSS_share_i and invokes IPA_{KZG} just as honest servers would. The simulator also runs MVBA faithfully, ensuring that any corrupted server $P_j^{l+1} \in \mathcal{A}^{l+1}$ receives a set of shares $\{[\langle c \rangle_i]_j^{l+1}\}_{i \in T}$ consistent with an agreed-upon subset $T \subseteq \mathcal{P}^l$, where $[\langle c \rangle_i]_j^{l+1} = f_{ci}(j)$.

Since the simulator follows the protocol specification exactly (from the viewpoint of any uncorrupted party), the adversary's view in **Game 1** is computationally indistinguishable from that in **Game 0**.

Game 2. Here, the simulator modifies how it provides shares to corrupted servers. Rather than genuinely invoking BACSS_share_i for each honest dealer P_i^l , the simulator takes as input the corrupted shares $\{[\langle c \rangle_i]_j^{l+1}\}_{i \in \mathcal{H}^l, j \in \mathcal{A}^{l+1}}$ that the adversary would see. It then randomly samples a degree- t polynomial $f'_c(\cdot)$. For each $P_i^l \in \mathcal{H}^l$, the simulator interpolates a polynomial $f'_{ci}(\cdot)$ consistent with these corrupted shares (so that $f'_{ci}(j) = [\langle c \rangle_i]_j^{l+1}$) and sets $f'_{ci}(0) = f'_c(i)$ to fix any remaining degrees of freedom.

From the adversary's perspective, this change does not alter the distribution of the shares it observes. The hiding property of the KZG with aggregation interfaces ensures that no additional information is revealed. Thus, the view of corrupted servers in **Game 2** remains indistinguishable from **Game 1**. Since **Game 1** is also indistinguishable from **Game 0**.

Game 3. This game is identical to **Game 2**, except that whenever an honest dealer P_i^l would run $\text{IPA}_{\text{KZG}}.\mathcal{P}$ to produce a proof for the public statement $(SP, \{C_{a_k}^l, C_{b_k}^l, C_{c_k}^{l+1}\}_{k \in [M]}, i_a, i_b, i_c, \mathcal{P}_{id})$, the simulator instead invokes the zero-knowledge simulator of the inner-product argument on the same public statement. By the perfect completeness of the inner-product argument, every simulated transcript for a true statement is accepted with non-negligible probability; by its statistical zero-knowledge property, the distribution of the simulated transcript is indistinguishable from that of a real transcript produced by $\text{IPA}_{\text{KZG}}.\mathcal{P}$. Hence the adversary's view in **Game 3** is indistinguishable from its view in **Game 2**.

Combining the indistinguishability of **Game 3** from **Game 2** with the previous transitions, we conclude that the adversary cannot distinguish the real execution **Game 0** from the simu-

lated one, which completes the secrecy proof for BatchMul. $\square$

APPENDIX E

SECURITY AND COMPLEXITY ANALYSIS OF Continuum

Theorem 4 (Privacy, and Guaranteed Output Delivery of Continuum). *Continuum securely realizes the desired properties of ADMPC (as in Definition 1).*

Proof. We prove that Continuum satisfies properties of ADMPC:

Privacy. During the input stage, secrecy is guaranteed by the underlying protocol BACSS. In the execution stage, secrecy is preserved separately for linear and multiplication gates: AggTrans (Lemma 4) ensures that the evaluation of linear gates does not leak any information beyond the corrupted parties' shares, while BatchMul (Lemma 8) guarantees the same for multiplication gates. In the output stage, clients broadcast shares to reconstruct the final outputs. Thus, throughout the entire execution, no additional information is leaked to an adversary beyond what could be inferred from the corrupted servers' own shares.

Guaranteed Output Delivery (GOD). During each epoch, circuit evaluation is divided into two parts. The computation phase is non-interactive and consists only of local evaluation, while the hand-off phase eventually terminates by the termination guarantees of AggTrans (Lemma 1) and BatchMul (Lemma 6). For linear gates, AggTrans ensures transfer consistency (Lemma 3), i.e., the transferred shares reconstruct to the same underlying secrets, preserving correctness across committees. For multiplication gates, BatchMul guarantees multiplication correctness (Lemma 7), ensuring that the multiplicative relations are faithfully preserved under resharing. Therefore, each hand-off eventually completes while preserving the correct output sharings across committees. The evaluation consequently progresses through all epochs, and all honest clients eventually reconstruct the correct final outputs. Thus, Continuum achieves guaranteed output delivery.

Combining these arguments, we conclude that Continuum satisfies privacy, and guaranteed output delivery as stipulated by Definition 1. $\square$

Complexity of Continuum. Consider an arithmetic circuit divided into D epochs, each of width W , consisting of λ linear gates and μ multiplication gates with $W = \lambda + \mu$. For each epoch, the communication cost is dominated by two sub-protocols: the asynchronous dynamic transfer of λ linear gates via AggTrans and the resharing of μ multiplication gates via BatchMul. The transfer of λ linear gates incurs a cost of $O(\kappa n^2 \cdot \lambda + \kappa n^3)$, while the resharing of μ multiplication gates requires $O(\kappa n^2 \cdot \mu + \kappa n^3)$. Summing these contributions, the total communication per epoch is

$$O(\kappa n^2 \cdot W + \kappa n^3).$$

Therefore, for a circuit of depth D , the overall communication complexity of Continuum is

$$O(\kappa n^2 \cdot WD + \kappa n^3 D).$$

F ADDITIONAL EVALUATION DETAILS AND RESULTS

A. MPC-based Shuffle Circuit Construction and Protocol Adaptations

In this appendix, we describe the shuffle circuit used in our evaluation, present the BatchRandBit protocol for generating secret-shared random sign bits in the ADMPC model, and explain how each switch layer maps to the computation and hand-off phases of our protocol.

1) *Shuffle Circuit Construction:* We adopt the iterated butterfly switching network from [5] as the shuffle benchmark. Figure 13 summarizes its two MPC programs: a switch that conditionally swaps two secret-shared inputs using one random sign bit, and a switching network consisting of $\log k$ butterfly iterations, each containing $\log k$ switch layers. Each layer applies $k/2$ switches to disjoint input pairs. With independent random sign bits, the resulting permutation is sampled from a nearly uniform distribution. For $k = 128$, the network has 49 switch layers, with 64 switches per layer.

MPC Program switch

- **Input:** $[i_1], [i_2]$
- **Output:** $[o_1], [o_2]$ which are i_1 and i_2 swapped with $1/2$ probability
- **Preprocessing:** random bit $[b]$, $b \in \{-1, 1\}$
- **Procedure:**

$$[c] := [b] \cdot ([i_1] - [i_2])$$

$$[o_1] := 2^{-1}([i_1] + [i_2] - [c])$$

$$[o_2] := 2^{-1}([i_1] + [i_2] + [c])$$

MPC Program switching-network

- **Input:** $[m_1], \dots, [m_k]$
- **Output:** $\pi(m_1, \dots, m_k)$, where π is sampled from a nearly uniform distribution
- **Procedure:**
 - for each of $\log k$ butterfly iterations, evaluate $\log k$ switch layers, where each layer applies $k/2$ switches to disjoint input pairs
 - finally, reconstruct the output of the final layer, $\text{Open}(\pi([m_1], \dots, [m_k]))$

Fig. 13. Permutation based on a switching network, adapted from [5].

2) *Secure Random Bit Generation in the ADMPC Model:* Generating a secret-shared random sign bit $[b]$ with $b \in \{-1, 1\}$ can be done via a standard technique, as applied in [5]: each server contributes a random field element $[r]$, computes $[s] \leftarrow [r]^2$ via a secure multiplication, publicly opens s , and sets $[b] \leftarrow [r]/\sqrt{s}$; if $s = 0$, the share is discarded, an event that occurs with negligible probability $1/q$. Since $\sqrt{s}$ is

a public value that reveals only $|r|$ and not the sign of r , the resulting $[b]$ is uniformly random in $\{-1, 1\}$.

To generate sign bits efficiently in the ADMPC model, we adapt the BatchRand structure from AD-MPC [8]: each server P_i^l acts as a dealer and distributes random field elements via BACSS; after MVBA selects $2t + 1$ completed dealers, each server P_j^{l+1} applies the Vandermonde matrix V to extract $t+1$ uniformly random sharings, then applies the bit-extraction procedure above to each sharing. We call this combined protocol BatchRandBit, presented as Algorithm 3.

3) *Mapping to the ADMPC Model*: The 49-epoch shuffle execution maps to the three-stage ADMPC structure as follows.

- **Input Stage.** Clients distribute 128 secret-shared inputs to the first committee $\mathcal{P}^1$ via BACSS. Clients also invoke BatchRandBit to supply the random sign bits required by $\mathcal{P}^1$ for the first switch layer.
- **Execution Stage.** Each switch layer $l \in [49]$ constitutes one epoch, evaluated by $\mathcal{P}^l$ through two sequential phases.
Computation phase. Each server in $\mathcal{P}^l$ evaluates the switch procedure from Figure 13 for each of the 64 switch gates. For the required multiplications, the committee invokes the intra-committee variant of BatchMul, presented as Algorithm 4.
Hand-off phase. The 128 switch outputs are transferred to $\mathcal{P}^{l+1}$ via AggTrans. Concurrently, committees $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$ jointly execute BatchRandBit to supply the random sign bits required by $\mathcal{P}^{l+1}$ for the next switch layer.
- **Output Stage.** After the 49th switch layer, the final committee $\mathcal{P}^{49}$ opens and releases the permuted outputs to the clients.

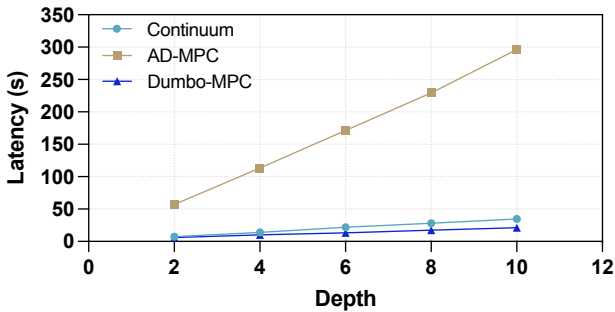


Fig. 14. Latency comparison of Continuum, AD-MPC, and Dumbo-MPC with circuit width $w = 100$, committee size $n = 16$, and varying circuit depth $d \in \{2, 4, 6, 8, 10\}$.

B. The Impact of Circuit Depth on Latency

We evaluate how circuit depth influences the latency of Continuum relative to AD-MPC and Dumbo-MPC. We use mixed circuits with a 1:1 ratio of multiplication to linear gates in each layer. We fix the circuit width $w = 100$ and the committee size $n = 16$, while varying the circuit depth over $d \in \{2, 4, 6, 8, 10\}$.

TABLE II
SYSTEM-WIDE COMMUNICATION OVERHEAD PER HAND-OFF. ALL VALUES ARE IN DECIMAL MB.

Protocol/component	Committee size n				
	4	10	16	22	128
RandGen	2.19	13.63	34.48	70.83	7,882.53
ADtrans	2.33	25.35	76.80	174.64	12,008.96
AD-MPC total	4.52	28.98	111.28	245.47	19,891.49
AggTrans-NoAgg	1.45	12.28	35.13	71.98	4,591.48
AggTrans	1.00	8.75	25.46	53.16	3,895.23

As shown in Figure 14, AD-MPC exhibits substantially higher latency than both Continuum and Dumbo-MPC across all evaluated depths. Continuum consistently outperforms AD-MPC while remaining close to Dumbo-MPC. As the circuit depth increases, the latency difference between Continuum and Dumbo-MPC becomes larger because Continuum changes committees between epochs, whereas Dumbo-MPC uses a static committee.

C. Communication and Computation Overheads of Transfer Protocols

This appendix complements the transfer-protocol evaluation in Figure 8 by providing detailed breakdowns of the communication and local cryptographic overheads measured for AD-MPC, AggTrans-NoAgg, and AggTrans.

Communication overhead. Table II reports the communication overhead per hand-off when transferring $B = 100$ sharings. Even after excluding RandGen from the AD-MPC pipeline, AggTrans-NoAgg reduces communication relative to ADtrans by 37.77%–61.77% across the evaluated committee sizes. This reduction comes from replacing ADtrans’s hbPolyCom-based transfer with commitment-anchored resharing instantiated using KZG [28]. The constant-size KZG evaluation proofs remove the $\log n$ factor from the B -dependent communication term, reducing it from $O(\kappa n^2 \log n \cdot B)$ to $O(\kappa n^2 \cdot B)$. The aggregation interfaces provide a further 15.16%–31.03% communication reduction over AggTrans-NoAgg. Specifically, for each dealer, the unaggregated variant contains B prior-epoch evaluation proofs, B evaluation proofs at point 0 for the resharing polynomials, and B corresponding evaluation commitments. As shown in Algorithm 1, the aggregation interfaces compress these three B -element vectors into W_i^l , $W_{i,0}^{l+1}$, and $T_{i,0}^{l+1}$, respectively, reducing the corresponding communication from $3B$ group elements to three. Together with the elimination of RandGen, these two improvements allow AggTrans to reduce the communication of the complete AD-MPC transfer pipeline by 77.12%–80.42%.

Computation overhead. Table III reports the per-server local cryptographic overhead per circuit layer when transferring a batch of $B = 100$ sharings. For ADtrans and AggTrans-NoAgg, the generation phase is dominated by constructing evaluation proofs for the resharing polynomials and has the same asymptotic order. The increasing computation gap be-

Algorithm 3 BatchRandBit for $\mathcal{P}^l$ and $\mathcal{P}^{l+1}$

Inputs: batch size B , Vandermonde matrix V of size $(t+1) \times (2t+1)$ over $\mathbb{Z}_q$, and other system parameters

Outputs: P_j^{l+1} obtains `rand_bitsj`, a set of secret-shared random sign bits

Server P_i^l (in committee $\mathcal{P}^l$):

// 1. Distribute random values via BACSS

```

101: sample  $B$  random secrets  $(r_{i,1}, \dots, r_{i,B}) \xleftarrow{\$} \mathbb{Z}_q$ 
102: for  $k \in [B]$ 
103:    $(C_{r_{ik}}^{l+1}, \hat{f}_{r_{ik}}^{l+1}(\cdot)) \leftarrow \text{PolyCom}(SP, f_{r_{ik}}^{l+1}(\cdot))$ 
   // s.t.  $\hat{f}_{r_{ik}}^{l+1}(0) = r_{i,k}$ 
104:   for  $j \in [n]$ 
105:      $w_{r_{ik},j}^{l+1} \leftarrow \text{ProveEval}(SP, j, f_{r_{ik}}^{l+1}(\cdot), \hat{f}_{r_{ik}}^{l+1}(\cdot))$ 
106: invoke BACSS_sharei as dealer to share  $(r_{i,1}, \dots, r_{i,B})$ 
   using KZG commitments  $\{C_{r_{ik}}^{l+1}\}_{k \in [B]}$ 

```

Server P_j^{l+1} (in committee $\mathcal{P}^{l+1}$):

// 2. Agree on $2t+1$ valid and completed dealers

```

201:  $T_j \leftarrow \emptyset$ ,  $\text{rand\_shares}_j \leftarrow \emptyset$ 
202: activate BACSS_sharei for every  $i \in [n]$ 
203: upon receiving  $\{([e_{i,k}]_j^{l+1}, C_{r_{ik}}^{l+1}, w_{r_{ik},j}^{l+1})\}_{k \in [B]}$  from
   BACSS_sharei

```

// Validity checked via VerifyEval

```

204:  $T_j \leftarrow T_j \cup \{i\}$ 
205:  $\text{rand\_shares}_j \leftarrow \text{rand\_shares}_j \cup \{[e_{i,k}]_j^{l+1}\}_{k \in [B]}$ 
206: if  $|T_j| = 2t+1$ 
207:   call MVBA with snapshot  $T_j$  as input
   // MVBA's predicate checks if  $T_j \supseteq \text{input}$ 
208: wait for MVBA output  $T$ , w.l.o.g.  $T = \{1, \dots, 2t+1\}$ 
// 3. Extract uniform random sharings via Vander-
monde matrix
209: for  $k \in [B]$ 
210:    $[r_{1,k}, \dots, r_{t+1,k}]_j^{l+1} \leftarrow V \cdot ([e_{1,k}, \dots, e_{2t+1,k}]_j^{l+1})^\top$ 
// 4. Public reconstruction and normalization
211:  $\{[s_{m,k}]_j^{l+1}\}_{m \in [t+1], k \in [B]} \leftarrow \text{Mul}(\{([r_{m,k}]_j^{l+1}, [r_{m,k}]_j^{l+1})\})$ 
212:  $\text{rand\_bits}_j \leftarrow \emptyset$ 
213: for  $m \in [t+1]$ 
214:   for  $k \in [B]$ 
215:      $s_{m,k} \leftarrow \text{Open}([s_{m,k}]_j^{l+1})$ 
216:     if  $s_{m,k} = 0$ : continue
217:      $\alpha_{m,k} \leftarrow \sqrt{s_{m,k}}$ 
218:      $[b_{m,k}]_j^{l+1} \leftarrow \alpha_{m,k}^{-1} \cdot [r_{m,k}]_j^{l+1}$ 
219:    $\text{rand\_bits}_j \leftarrow \text{rand\_bits}_j \cup \{[b_{m,k}]_j^{l+1}\}$ 
220: return  $\text{rand\_bits}_j$ 

```

TABLE III

PER-SERVER LOCAL CRYPTOGRAPHIC OVERHEAD PER CIRCUIT LAYER.
ALL VALUES ARE IN SECONDS.

Protocol/component	Committee size n				
	4	10	16	22	128
RandGen	0.81	1.84	2.78	3.49	17.69
ADtrans	0.65	2.42	4.05	7.56	83.64
AD-MPC total	1.46	4.26	6.83	11.05	101.33
AggTrans-NoAgg	1.09	2.41	3.80	5.20	33.29
AggTrans	0.36	0.64	1.01	1.33	12.73

tween these two protocols is consistent with the verifier complexities summarized in Table I: AD-MPC's hbPolyCom -based verification requires $O(Bn \log n)$ computation, whereas the KZG-based verification in our design requires $O(Bn)$. AggTrans-NoAgg individually verifies B prior-epoch anchoring proofs and B fresh point-zero proofs for each dealer, requiring $2B$ pairing checks in total. The aggregation interfaces compress each set of B proofs into a single aggregated proof, reducing the anchoring verification from $2B$ pairing checks to two. Consequently, AggTrans reduces local cryptographic overhead by 61.76%–74.42% relative to AggTrans-NoAgg.

D. Communication and Computation Overheads of Multiplication Protocols

This appendix complements the multiplication-protocol evaluation in Figure 9 by providing detailed breakdowns of the communication and local cryptographic overheads of AD-MPC, BGW-AggTrans, and BatchMul.

Communication overhead. Table IV reports the communication overhead per multiplication layer with $M = 100$ multiplication gates. Compared with AD-MPC, BGW-AggTrans reduces communication by 82.12%–89.27% across $n \in \{4, 10, 16, 22\}$. This improvement primarily comes from replacing AD-MPC's offline/online paradigm with BGW-style multiplication, thereby eliminating the epoch-by-epoch preprocessing required to generate Beaver triples. In particular, AD-prep alone accounts for 73.30%–82.74% of AD-MPC's total communication. BGW-AggTrans also eliminates RandGen and replaces ADtrans with AggTrans during the hand-off phase. As shown in Table II, AggTrans communicates 57.08%–69.56% less data than ADtrans, further widening the gap between the two constructions. Our BatchMul protocol further reduces communication by 42.53%–48.32% relative to BGW-AggTrans by encoding the M product relations in a single $O(\log M)$ -size argument per dealer and combining degree reduction and resharing within a single hand-off. Together, these optimizations allow BatchMul to reduce communication by 89.73%–94.22% relative to AD-MPC.

Computation overhead. Table V reports the per-server local

Algorithm 4 Intra-committee BatchMul for $\mathcal{P}^l$

Public Inputs: KZG commitments $\{C_{a_k}^l, C_{b_k}^l\}_{k \in [M]}$ and other system parameters

Private Inputs: P_i^l holds degree- t shares $\{[a_k]_i^l, [b_k]_i^l\}_{k \in [M]}$ with randomness $\{\hat{a}_k]_i^l, \hat{b}_k]_i^l\}_{k \in [M]}$, evaluation proofs $\{w_{a_k i}^l, w_{b_k i}^l\}_{k \in [M]}$, and degree- $2t$ product shares $\{\langle c_k \rangle_i^l\}_{k \in [M]}$ with randomness $\{\hat{c}_k]_i^l\}_{k \in [M]}$

Public Outputs: New KZG commitments $\{C_{c_k}^l\}_{k \in [M]}$

Private Outputs: P_i^l obtains $\{[c_k]_i^l\}_{k \in [M]}$ with randomness $\{\hat{c}_k]_i^l\}_{k \in [M]}$ and evaluation proofs $\{w_{c_k i}^l\}_{k \in [M]}$

Server P_i^l (in committee $\mathcal{P}^l$):

// 1. Prove product relations and reshare

```

101: for  $k \in [M]$ 
102:    $(C_{c_k i}^l, \hat{f}_{c_k i}^l(\cdot)) \leftarrow \text{PolyCom}(SP, f_{c_k i}^l(\cdot))$ 
      // s.t.  $\hat{f}_{c_k i}^l(0) = \langle c_k \rangle_i^l, \hat{f}_{c_k i}^l(0) = \langle \hat{c}_k \rangle_i^l$ 
103:   for  $j \in [0, n]$ 
104:      $w_{c_k i, j}^l \leftarrow \text{ProveEval}(SP, j, f_{c_k i}^l(\cdot), \hat{f}_{c_k i}^l(\cdot))$ 
105:  $\text{proof}_i \leftarrow \text{IPA}_{\text{KZG}}.\mathcal{P} \left( \left\{ \begin{array}{l} (C_{a_k}^l, i, [a_k]_i^l, [\hat{a}_k]_i^l, w_{a_k i}^l), \\ (C_{b_k}^l, i, [b_k]_i^l, [\hat{b}_k]_i^l, w_{b_k i}^l), \\ (C_{c_k i}^l, 0, \langle c_k \rangle_i^l, \langle \hat{c}_k \rangle_i^l, w_{c_k i, 0}^l) \end{array} \right\} \right)$ 
      for  $k \in [M]$ 
106: invoke  $\text{BACSS\_share}_i$  as dealer to reshare  $\{\langle c_k \rangle_i^l\}_{k \in [M]}$ 

```

using KZG commitments $\{C_{c_k i}^l\}_{k \in [M]}$ with broadcasting proof proof_i

// 2. Verify multiplication correctness via IPA_{KZG}

```

107:  $T_i \leftarrow \emptyset, \text{mul\_shares}_i \leftarrow \emptyset$ 
108: activate  $\text{BACSS\_share}_j$  instance for every  $j \in [n]$ 
109: upon receiving valid shares  $\{(\langle c_k \rangle_j^l, [\hat{c}_k]_j^l, C_{c_k j}^l, w_{c_k j, i}^l)\}_{k \in [M]}$ 
      and proof  $\text{proof}_j$  from  $\text{BACSS\_share}_j$ 
      // Validity verified via  $\text{VerifyEval}$ 

```

```

110:   if  $\text{IPA}_{\text{KZG}}.\mathcal{V} \left( \left\{ \begin{array}{l} (C_{a_k}^l, j), \\ (C_{b_k}^l, j), \\ (C_{c_k j}^l, 0) \end{array} \right\}_{k \in [M]}, \text{proof}_j \right) = 1$ 
111:      $T_i \leftarrow T_i \cup \{j\}$ 

```

// 3. Agree on a common subset via MVBA

```

112:   if  $|T_i| = 2t + 1$ 
113:     call MVBA with snapshot  $T_i$  as input
      // MVBA's predicate checks if  $T_i \supseteq \text{input}$ 
114:   wait for MVBA output  $T$ , w.l.o.g.  $T = \{P_1^l, \dots, P_{2t+1}^l\}$ 

```

// 4. Interpolate degree- t shares

```

115:   for  $k \in [M]$ 
116:      $[c_k]_i^l \leftarrow \sum_{j=1}^{2t+1} \lambda_j [\langle c_k \rangle_j^l]$ 
      //  $\lambda_j$  is the Lagrange coeff. for eval at 0
117:    $\text{mul\_shares} \leftarrow \text{mul\_shares} \cup \{[c_k]_i^l\}_{k \in [M]}$ 
118:   return  $\text{mul\_shares}$ 

```

TABLE IV

SYSTEM-WIDE COMMUNICATION OVERHEAD PER MULTIPLICATION LAYER. ALL VALUES ARE IN DECIMAL MB.

Protocol/component	Committee size n				
	4	10	16	22	128
ADprep	12.63	158.07	506.14	1,197.93	–
RandGen	2.19	13.63	34.48	70.83	–
Exec	0.08	0.72	2.14	4.49	–
ADtrans	2.33	25.35	76.80	174.64	–
AD-MPC total	17.23	197.77	619.56	1,447.89	–
BGW-AggTrans	3.08	27.04	76.24	155.29	9,147.65
BatchMul	1.77	14.78	41.34	83.71	4,727.29

cryptographic overhead per multiplication layer with $M = 100$ multiplication gates. Exec is excluded because it does not invoke the instrumented commitment or proof operations. BGW-AggTrans reduces local cryptographic overhead by 52.52%–82.51% relative to AD-MPC across $n \in \{4, 10, 16, 22\}$, primarily by eliminating ADprep, which accounts for 74.96%–84.00% of AD-MPC's overhead. It also eliminates RandGen and replaces ADtrans with AggTrans. As shown in Table III, the AggTrans component accounts for 10.74%–15.01% of BGW-AggTrans's local cryptographic overhead, while BGW degree reduction accounts for the remaining 84.99%–89.26%, primarily because each server verifies the standalone evalua-

TABLE V

PER-SERVER LOCAL CRYPTOGRAPHIC OVERHEAD PER CIRCUIT LAYER. ALL VALUES ARE IN SECONDS.

Protocol/component	Committee size n				
	4	10	16	22	128
ADprep	4.31	20.06	37.58	59.15	–
RandGen	0.85	1.90	2.80	3.65	–
ADtrans	0.59	2.44	4.36	8.00	–
AD-MPC total	5.75	24.40	44.74	70.80	–
BGW-AggTrans	2.73	5.87	9.09	12.38	84.80
BatchMul	1.14	1.96	2.71	3.63	23.48

tion and product-relation proofs from all n dealers.

BatchMul further reduces local cryptographic overhead by 58.24%–72.31% relative to BGW-AggTrans. This reduction comes from replacing the $O(M)$ standalone KZG evaluation-proof and product-relation checks performed for each dealer with a constant number of aggregated KZG checks and a single inner-product-argument verification. Moreover, deferring verifiable degree reduction to the hand-off phase eliminates the separate proof-generation and verification workload of the BGW computation phase. Together, these optimizations allow BatchMul to reduce local cryptographic overhead by 80.17%–94.87% relative to AD-MPC.