# Eras MPC: Granular Preprocessing and Abort Recovery with Dynamic Committees

Ivan Damgard[1], Bernardo David[2⋆] and Rahul Satish[2⋆]

[1] Aarhus University
ivan@cs.au.dk
[2] IT University of Copenhagen
bernardo@bmdavid.com, rahs@itu.dk

**Abstract.** MPC with dynamic committees achieves scalability by having small committees execute each round of the computation, instead of involving all parties in the entire execution. However, achieving Guaranteed Output Delivery in efficient information theoretical protocols requires larger honest majority committees, limiting scalability. Moreover, using preprocessed correlated randomness to speed up the computation is prohibitively expensive due to the cost of transferring such randomness to parties that have not yet joined the execution.
We propose Eras MPC, a model where small dishonest majority computation committees execute one or more rounds of the computation with the help of honest majority preprocessing and checkpointing committees. The computation is carried out in *Eras*, which are sets of shorter epochs. In the beginning of each Era's first epoch, a preprocessing committee generates correlated randomness for the computation epochs to be executed by different computation committees. In the end of an Era, checkpoint committees verify the current state of the computation, save this state if it is valid and provide the latest valid state to the next Era.
We show an Eras MPC protocol where the preprocessing is carried out using an honest majority MPC protocol to generate authenticated Beaver triples, requiring only enough rounds of interaction to perform two subsequent batches of multiplications. The computation is done via the SPDZ online phase executed in each epoch by a small dishonest majority committee. Finally, intermediate state is checkpointed by verifying the validity of wire shares and saving wire values in 4 rounds, each executed by a different honest majority checkpoint committee.

## 1 Introduction

Multiparty computation (MPC) allows mutually distrustful parties to evaluate a function of their private inputs while revealing only the prescribed output [8, 24, 45]. Classical protocols typically require a fixed set of computation parties to remain available throughout the execution. For long computations, this requirement limits participation and makes completion depend on the continued availability of the same parties. MPC with dynamic committees [1–4, 11–13, 17, 25, 30, 33, 34, 40, 43, 49, 55] addresses this restriction.

In dynamic MPC models, the execution is distributed across successive committees, each of which evaluates part of the circuit and transfers shares of the intermediate wire values to its successor. In Fluid MPC, a committee may join at any point of the execution and execute any number of rounds, communicating only with the immediately next committee. In YOSO MPC, each round is executed by a different anonymous committee selected at random, who may communicate with any future anonymous committee via ideal channels. What they all have in common is that each committee executes what it is assigned to do and passes the private computation state to a future committee, allowing the computation to continue as its participants change.

This flexibility of participation comes with a recurring cost. A new committee must obtain the intermediate results needed to continue, while preserving their privacy and correctness. Transferring this state requires communication in addition to that used for the computation itself. Both the number of parties involved and the frequency of committee changes therefore matter for efficiency. Allowing each committee to execute several rounds reduces the frequency of transfers, as demonstrated by recent work [33]. Reducing committee sizes offers a complementary opportunity, but depends on the corruption threshold that the protocol requires each committee to satisfy.

⋆ Funded by the Independent Research Fund Denmark (IRFD) grant number 0165-00079B and the Swiss National Science Foundation grant number 10009029.

Existing protocols for MPC with dynamic committees hold every committee to the same corruption threshold, with one exception that relaxes it for all but one committee at the cost of evaluating the circuit homomorphically [11]. The consequences at the two ends of that choice are opposite.

At the honest-majority end are the expensive protocols achieving guaranteed output delivery (GOD), which guarantees that all honest parties receive the output of the computation. The best maximally fluid protocols with GOD have communication complexity of $O(n^9)$ [30] and $O(\kappa n^{18})$ [34] for perfect and statistical security, respectively. Obtaining GOD in the YOSO model via the CDN approach [17, 26, 43] requires a threshold decryption to compute every multiplication gate, placing expensive public-key operations in the online phase.

At the dishonest-majority end are the fast protocols. A committee tolerating $t < n$ corruptions need contain only one honest party, so it can be smaller, and SPDZ-style [28, 29] evaluation over such a committee has local additions and a single broadcast round per layer of multiplications. These protocols achieve security with abort. With dynamic committees an abort loses the whole computation, since the input parties have left, the preprocessing is consumed, and the state can only be recovered if all parties are honest.

We seek to retain the efficiency of dishonest-majority online computation while ensuring to not have to restart a computation from scratch when an abort occurs. The latter requires keeping a verified private state of intermediate computations. Having access to such a state would allow new committees to retry the unfinished work and, if optimistic attempts keep failing, provide the input to a protocol with GOD. The central question is:

*Can dynamic MPC retain the efficiency of dishonest-majority online computation while keeping verified progress and supporting a fallback to GOD?*

We answer this question affirmatively with *Eras MPC*, which separates the committees performing optimistic computation from those that verify and retain the intermediate state that enables recovery from aborts. The online phase can then use a dishonest-majority protocol, while honest-majority committees support preprocessing, verification, and an optional GOD fallback when needed.

### 1.1 Our Results

We present four main contributions: We first formalize the execution model, also presenting a protocol that realizes it. We then show optimizations and variants extending the basic construction. Finally, we provide communication lower bounds that motivate the model's key design choices. Each is summarized below.

**Eras MPC Model.** We define Eras MPC, a model where the execution is divided in Eras comprised of multiple epochs, which in turn are comprised of multiple rounds. An *era* starts with a preprocessing epoch that generates correlated randomness for $m$ consecutive computation epochs executed by different committees, each evaluating an assigned sub-circuit. An era ends with a checkpointing phase that verifies the resulting intermediate state, retains the latest valid state and provides inputs for the next era. This later phase is executed in a maximally fluid manner, *i.e.*, each round is executed by a different checkpoint committee. The model assigns preprocessing, computation, and checkpointing to separate committees. This separation allows for using larger honest-majority committees only for periodic preprocessing and checkpointing, while the circuit itself is evaluated by smaller dishonest-majority computation committees. The committees assigned to each era are taken from publicly ordered pools. If the intermediate computation state of an era is successfully verified in the checkpoint phase, the next unused committees are assigned to the next era; an unsuccessful verification causes the same era to be re-executed by the next unused committees. The ideal functionality $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ formally defines the computations, transitions and the conditions under which the available pools are exhausted.

**Eras MPC Protocol.** We construct Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$ for arithmetic circuits over $\mathbb{F}_p$ and prove that it realizes $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ in the universally composable (UC) framework. Theorem 1 establishes security against a static adversary corrupting a majority of the parties in computation committees and a minority of the parties in preprocessing and checkpoint committees. Our construction assumes

functionalities for preprocessing (which we later construct), commitments and linearly homomorphic threshold encryption as setup.

We realize preprocessing through honest-majority arithmetic MPC (Section 3.4), which generates authenticated multiplicative triples delivered by the preprocessing committee directly to the computation committees. Moreover, preprocessing generates correlated randomness used to reshare intermediate state among computation committees and verify the validity (encrypted) state output by the last computation committee. The benefit is that this constant round protocol is executed by each era's preprocessing committee, while computation is done by smaller dishonest-majority committees.

Given this preprocessing, each computation committee evaluates their sub-circuit using the SPDZ [29] online phase. As in SPDZ, additions are local, and each multiplication layer requires a round of communication. Computation committees within an era transfer their state to the next computation committee using preprocessed data. The last computation committee sends encrypted versions of their authenticated wire value shares to the first checkpoint committee in the era, which also receives the public information for checking multiplications.

At the end of an era, the checkpoint protocol uses four successive honest-majority committees to verify the era's intermediate state in encrypted form, and, if it is valid, store it and redistribute it as fresh SPDZ shares towards the first computation committee in the next era. If the state is invalid, the checkpoint committee rehares the previously valid state to the next computation committee. Therefore, an accepted checkpoint always provides a private state that can be used independently of the computation committees that produced it. A failed verification only repeats the computation of the current era, as the previously valid state remains available. An initial checkpoint of the inputs provides the same protection before the first era succeeds.

This protocol provides security with abort if its pools are exhausted. The GOD extension in Section 4 reserves the committees and communication resources needed for a fallback. If not enough committees are available to retry the optimistic computation, checkpoint committees supply robust shares of the last verified state as private inputs to a protocol with GOD. That protocol completes the remaining sub-circuits under its own corruption and communication assumptions. This is what turns preserved progress into a path to completion: the fallback has an available, verified state, even if every optimistic attempt is disrupted. All this could happen even after the original input parties and previous computation committees go offline.

**Optimizations and Variants.** The retained state also supports extensions beyond recovery. Section 4 describes these variants. First, using a public bulletin board, a single preprocessing committee can generate correlated randomness for multiple future eras without resharing, enabling more flexible deployment and public verifiability of intermediate computation states. Second, the protocol extends naturally to *reactive computations*, where era boundaries serve as input/output points and sub-circuits can be chosen adaptively based on revealed outputs. Third, we describe a MAC key rotation mechanism that achieves *post-compromise integrity without erasures*: if a committee is corrupted after its execution has ended, the integrity of all subsequent eras is preserved by rotating the MAC key during the era-boundary relay. Finally, the checkpoint state provides a natural *fallback for Guaranteed Output Delivery*: if the dishonest-majority computation pool is exhausted, checkpoint committees can decrypt the latest verified state and hand it to any protocol with GOD, without restarting from scratch.

**Communication Lower Bounds.** The model's central choice is to let a computation committee act for several rounds, so that state is transferred only once a computation epoch ends, rather than at every round. Moreover, this transfer is made cheaper by preprocessed randomness, while the more expensive resharing between eras happens only once per era. We give evidence that this choice is necessary, by showing that the cost of a state transfer via resharing without preprocessed randomness is inherent. Any such transfer of a single wire value between disjoint committees of sizes $n$ and $n'$ at the optimal threshold requires $\Omega(n \cdot n' \cdot |w|)$ communication (Section 5). This bound highlights two costs in such dynamic models: The communication grows with the committee sizes, and with the number of state transfers. So the cost of transferring state falls either by shrinking the committees between which it moves, or by moving it less often. Eras MPC does both, and does them independently: the threshold split fixes how large a computation committee has to be, and the computation epoch length fixes how often a state transfer is done.

| Protocol | Honest maj. needed in | Rds. per comm. | Guarantee | PK ops, online | Loss on failure |
|---|---|---|---|---|---|
| Fluid MPC [25] | all comm. | 1 | abort | none | all |
| Layered MPC [30] | all ($t < n/3$) | 1 | GOD | none | n/a |
| Layered MPC [34] | all comm. | 1 | GOD | none | n/a |
| [12, 13] | all comm. | 1 | abort | none | all |
| YOSO-CDN [17] | all comm. | 1 | GOD | dec./mult. | n/a |
| YOSO-FHE [11] | one comm. | 1 | GOD | FHE/gate | n/a |
| Rumors MPC [33] | all comm. | $O(1)$ | GOD | none | n/a |
| Le Mans [55] | none† | $O(1)$ | abort | none | all |
| **Eras MPC** | prep., checkpt. | $O(1)$ | abort → GOD‡ | enc./epoch | one era |

**Table 1.** Positioning against dynamic-committee MPC protocols. Guarantees hold under each protocol's own corruption and setup assumptions. Rounds are counted for the committees that evaluate the circuit, and public-key (PK) operations on the online path only. "Loss on failure" is the committed computation destroyed by one deviating party. †Requires every party that will ever act to join a pseudorandom correlation generator setup beforehand. ‡Abort optimistically; GOD by handing the last checkpoint to an honest-majority fallback that evaluates only the residual circuit.

**Efficiency and Parameters.** Translating computation committee size, computation epoch length and era length, into concrete protocol parameters reveals the precise efficiency gains of our model. As established, the corruption threshold assigned to a specific role strictly dictates its required committee size, while the epoch and era lengths dictate the frequency of state transfers. To these constraints, our model adds a third, operational parameter: a retry budget, which limits the number of unsuccessful optimistic attempts permitted before invoking the guaranteed fallback. Below, we quantify these choices and their impact on complexity; Section 3.5 then details the final concrete costs.

*Committee size and corruption thresholds.* One of the main motivations for protocols with dynamic committees is sampling small committees at random from a larger set of parties. This allows for executing each round of the protocol with a smaller committee, instead of requiring all parties to execute all rounds. It is known [9, 14, 31, 32] that guaranteeing that a random committee has at most a certain number of corrupted parties except with negligible probability implies that obtaining honest-majority requires sampling much larger committees than obtaining dishonest-majority. For example, when sampling committees from a set of $n = 10000$ parties where $t = 0.3n$ are corrupted such that a prescribed corruption ratio is guaranteed except with probability $2^{-60}$, if the identities of parties in a sampled committee are publicly known, guaranteeing honest-majority requires committees of 462 parties, while guaranteeing dishonest-majority requires committees of only 35 parties [32]. In the same scenario but keeping the committee anonymous until they act (*e.g.* the YOSO model), guaranteeing honest-majority requires committees of 1450 parties, while guaranteeing dishonest-majority requires committees of only 210 parties [31].

*Epoch and era lengths.* The frequency of committee changes is set independently of committee size. For fixed sizes, let an epoch evaluate a subcircuit of depth at most $\delta$, and let an era contain $m$ epochs. Increasing $\delta$ reduces how often state is transferred, but it requires each computation committee to remain online for a longer duration. Conversely, increasing $m$ reduces how often an expensive era boundary checkpoint is executed, without changing how long any individual computation committee must participate. However, increasing $m$ also places more computation between checkpoints, meaning more work must be repeated if a failure occurs. This is the core trade-off between verification overhead and repeated work that the model leaves to the instantiation.

Ultimately, committee size, era length, and the retry budget, express deterministic costs and guarantees. Because the retry budget strictly bounds the number of unsuccessful attempts before the fallback is invoked, none of our efficiency claims require a probabilistic corruption model, nor do they rely on the assumption that a randomly sampled fresh committee will eventually cooperate.

**Comparison with dynamic MPC.** Fluid MPC [25] and YOSO MPC [43] established models for MPC with dynamic committees, and Layered MPC [30] restricts communication to the immediately following committee. Rumors MPC [33] studies the Multi-Layered model, allowing constant-round interaction within a committee with one round to transfer state to its successor. Le Mans [55] achieves

dynamic dishonest-majority SPDZ-style computation with abort, using universal preprocessing generated among the parties that may later take part. Bhadauria et al. [11] obtain GOD with only one honest-majority committee, using threshold fully homomorphic encryption (FHE) and public computation. Table 1 places our model among these. This comparison identifies the combination our construction must support: inexpensive online phase, preprocessing by successive committees, and verified state that remains available for recovery and fallback. We next explain why this combination requires more than periodic verification, and how the protocol achieves it.

### 1.2 Why Natural Approaches Fall Short.

To motivate our design choices, we first examine why existing techniques are insufficient on their own to achieve both inexpensive online phase computation and guaranteed recovery. Several familiar techniques supply parts of this combination. Setting out what each leaves undone identifies the exact requirements our checkpointing mechanism must meet.

**Run SPDZ with changing committees.** Dynamic SPDZ supplies the desired online threshold and arithmetic, and universal preprocessing, generated in advance among every party that may later come online, is one way to supply the correlations it consumes. For an execution organized around successive preprocessing committees the distribution problem differs: correlations must be produced for particular computation roles, delivered to their holders, and replenished after an aborted attempt. Carrying an inventory of unused triples along with the live state would make each committee change depend on future work. Solving delivery alone also leaves the abort problem intact, since a committee holding additive shares can still prevent their reconstruction by withholding its contribution.

**Check the computation more often.** A periodic MAC check detects inconsistency sooner and limits how much is computed before detection. It does not leave the state in a form that survives a computation party's refusal to continue. Even after a successful check of earlier openings, the current wires remain additively shared among that same committee, and a later committee cannot reconstruct the missing shares from the fact that a check succeeded. A useful checkpoint must therefore validate the intermediate state as well as the preceding openings, and establish an independent means of transferring it, while the previous checkpoint remains available until the candidate is accepted.

**Eliminate cheaters.** In a fixed population with an honest majority, player elimination [48] localizes a fault to a pair of parties containing at least one corrupt party, discards both, and reruns the affected segment. None of the prerequisites for this approach hold in our setting. It requires the survivors to retain an honest majority, whereas a computation committee may hold a single honest party, as likely to be discarded as the party accused alongside them. It requires a persistent set to eliminate from, whereas our committees act once and disperse. It requires a localizable fault, whereas a failed batched check is a global zero test naming no party; making it name one requires per share commitments from every computation party, which is the online cost the deferred check exists to avoid. Finally, the per committee bound permits a corrupt party in every computation committee even when memberships are disjoint, so replacing a failed committee does not imply its replacement has fewer corrupt parties or will complete.

**Invoke a GOD protocol after an abort.** A GOD protocol needs private inputs that correctly describe the computation it is to finish. Invoked on the original inputs, those inputs must still be available and the whole circuit may be repeated. Invoked on intermediate values, their correctness and availability must already be established. Asking the failed committee to supply them cannot resolve an abort caused by that committee's silence, and encrypting whatever shares it chooses to submit does not establish that the encrypted state is correct. The transition is possible in Eras MPC only because a verified state has already been retained outside the computation committee.

**Use honest majority or homomorphic evaluation throughout.** An honest majority protocol is robust throughout, and allowing several rounds per committee substantially reduces handoff frequency, as in the Multi Layered model. It retains the honest majority condition on the committees

performing the evaluation, so no committee shrinks and the handoff cost that dominates there is unchanged. Threshold FHE constructions reduce honest majority participation to a single committee, so reducing that participation is not by itself what distinguishes our construction; the distinction is where the encryption is used. CDN style constructions also rely on threshold LHE but perform public key operations during multiplication, so the computation proceeds at public key speed. A proof based alternative, in which each committee certifies its subcircuit by verifiable computation, pays a per gate proving cost and yields a proof rather than a state, and so does not make the computation available to anyone else, which a retry and a fallback both require.

### 1.3 Technical Overview

We now outline the core mechanics of our Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$, following the protocol's structure from preprocessing through computation and the era boundary checkpoint relay. One invariant holds across boundaries: after a successful boundary the retained ciphertexts encode the correct intermediate state and can supply fresh private shares to any subsequent committee, and after a failed one the previous checkpoint remains usable. In Appendix 1.4, we discuss exactly why current techniques fall short of achieving our results, as well as other related dynamic MPC models.

**Two Properties of the SPDZ2 Batch Check.** Our protocol builds on the SPDZ framework [28, 29], in which wire values are represented as authenticated additive sharings $\langle x \rangle = \big([x], [\alpha \cdot x]\big)$ under a global MAC key $\alpha \in \mathbb{F}_p$. The SPDZ2 batch-verification technique [28] reduces MAC consistency of all opened values to a single random-linear-combination test: for opened values $\{o^j\}$ with per-party MAC tag shares $\{\gamma_l^j\}$, one checks that $c := \sum_j \Big(-\alpha \cdot o^j + \sum_l \gamma_l^j\Big) = 0$, which holds if all tags are consistent with $\alpha$ and fails with overwhelming probability otherwise. Two properties of this check carry the construction. The first is standard; the second has not, to our knowledge, been used in this way.

The first is that the check is *stateless with respect to the computation*: it mentions only the opened values and the tag shares that accompanied them, and no intermediate wire, gate, or ordering. It can therefore be evaluated once, arbitrarily long after the openings, over an entire era's transcript, and it enforces the integrity of every opening in that era at once. Computation committees accumulate an unauthenticated transcript throughout the era without verifying any MAC tags, and the full transcript is validated in one pass at the era boundary.

The second is that the check is *linear in the key*: as a function of $\alpha$ it is affine with public coefficients, $c = \big(-\sum_j o^j\big) \cdot \alpha + \sum_{j,l} \gamma_l^j$. It can therefore be evaluated inside a linearly homomorphic ciphertext by parties holding an encryption of $\alpha$, so the verifier can be a fresh committee that has seen nothing of the era and holds no key material in the clear, and no fully homomorphic evaluation is needed.

Neither property suffices alone. Statelessness without linearity in the key would require the verifying committee to learn $\alpha$, at which point it could forge the computation it verifies. Linearity without statelessness would require it to hold the era's intermediate state, which a dishonest-majority computation committee cannot be asked to surrender intact. Together they let a small committee, in which all but one party may be corrupt, compute optimistically while a separate honest-majority committee decides afterwards whether it did so honestly.

**Preprocessing via Honest-Majority MPC.** The global MAC key $\alpha$ is coin tossed inside a threshold LHE [26] ciphertext by initial checkpoint committees. Considering a preprocessing committee with $n_{\mathsf{Prep}}$ parties and $t_{\mathsf{Prep}}$ corrupted parties, checkpoint committees also coin toss encrypted random coefficients of a degree-$t_{\mathsf{Prep}}$ polynomial $f$ with $f(0) = \alpha$. Evaluating $f$ at $1, \ldots, n_{\mathsf{Prep}}$ is linear on ciphertexts, so a checkpoint committee can threshold-decrypt the Shamir shares $\alpha_k = f(k)$ towards each party of the next preprocessing committee; by $t_{\mathsf{Prep}}$-secrecy no $t_{\mathsf{Prep}}$ corrupt preprocessing parties learn $\alpha = f(0)$. Fresh coefficients are sampled for every era, so the shares the adversary holds in one era say nothing about those it holds in another.

Each preprocessing committee runs an honest-majority MPC protocol $\mathcal{F}_{\mathsf{AMPC}}$ on a single fixed arithmetic circuit $\mathbf{C}_{\mathsf{Comp}}$ to produce the correlated randomness consumed by the computation committees of its era. The protocol must provide guaranteed output delivery, so that a corrupt preprocessing party cannot stall the era before it begins. The circuit samples random values $a, b$ together with

the intra-era handoff masks, and evaluates two sequential batches of multiplications: first $c = a \cdot b$, producing the Beaver triple product, then the MAC tags $\alpha \cdot a$, $\alpha \cdot b$, $\alpha \cdot c$. Two batches of parallel multiplications are necessary as well as sufficient, since the tag $\alpha \cdot c$ is a degree-three function of the sampled $a, b, \alpha$ and cannot be produced in one round of multiplications. Any information-theoretically secure honest-majority protocol such as [33, 38, 39, 46] realizes $\mathcal{F}_{\mathsf{AMPC}}$.

Preprocessing also produces authenticated random values in encrypted form used by the era's checkpoint committees to reshare the latest valid state. Each preprocessing party publishes $\mathsf{Enc}(\mathsf{pk}, r_i)$ together with a rerandomized $r_i \boxdot \mathsf{ct}_\alpha$ and an argument that the two share the witness $r_i$, and the aggregate over the verifying contributions satisfies $\gamma(r) = \alpha \cdot r$ by construction. Aggregating encrypted shares under a plaintext-knowledge argument alone would not establish this, since such an argument binds a party to $a$ value rather than to its own.

The key efficiency property is that no resharing is needed after preprocessing. Outputs of $\mathcal{F}_{\mathsf{AMPC}}$ are already robust secret shares distributed among preprocessing committee members, and each computation party reconstructs only its own share from messages addressed to it. Correlations therefore reach the committees that consume them in a single hop, rather than travelling through every committee that acts in between, and a later preprocessing committee can produce a fresh batch for a retried era without earlier computation parties returning.

**Computation and Intra-Era State Transfers.** Computation committees evaluate sub-circuits using the standard SPDZ online protocol. Addition gates are free; each multiplication gate consumes one Beaver triple and requires a single broadcast round to open $\epsilon = x - a$ and $\rho = y - b$, yielding $\langle x \cdot y \rangle = \langle c \rangle + \epsilon\langle b \rangle + \rho\langle a \rangle + \epsilon\rho\langle 1 \rangle$ by local computation. Per-party MAC tag shares for each opened value are stored locally but not yet verified.

At the end of an epoch, the computation committee compresses the values needed for verifying the intermediate state rather than forwarding them. A random public challenge is generated, and each party then encrypts the single field element obtained by combining its MAC tag shares under that challenge, attaches a NIZK proof of knowledge, and sends it to the era's first checkpoint committee. This costs one ciphertext and one NIZK per party per epoch, whatever the number of gates, after which the committee may leave: its individual shares and public openings need not be available to the later verifier.

At intra-era sub-circuit boundaries, committee $\mathcal{C}_{i,j}^{\mathsf{Comp}}$ passes output wires to $\mathcal{C}_{i,j+1}^{\mathsf{Comp}}$ via a masked broadcast. The preprocessing committee supplies two *independent* random values $\beta_k$ and $\eta_k$, each additively shared once to $\mathcal{C}_{i,j}^{\mathsf{Comp}}$ and once to $\mathcal{C}_{i,j+1}^{\mathsf{Comp}}$, so each party of $\mathcal{C}_{i,j}^{\mathsf{Comp}}$ broadcasts its share of $y_k$ masked by $\beta_k$ and its share of $\gamma(y_k)$ masked by $\eta_k$, and the party of the same index in $\mathcal{C}_{i,j+1}^{\mathsf{Comp}}$ subtracts its own shares of $\beta_k, \eta_k$. Because the two committees hold independent sharings of each mask, masks cancel in the aggregate and the receiving committee ends with a fresh authenticated sharing of the same wire, no share of which is ever reconstructed. Since $\beta_k$ and $\eta_k$ are uniform and independent, the broadcast is an information-theoretic one-time pad revealing neither $y_k$ nor $\alpha$; masking the tag with $\alpha\beta_k$ instead would make the broadcast value and tag a consistent pair and expose $\alpha$. Independence also means secrecy does not depend on who receives the values, so a broadcast suffices and no threshold decryption or proof of correctness is needed. A computation committee that deviates here deviates on an authenticated value and is caught by the checkpoint committee.

**The End of Era Checkpoint.** At the end of era $i$, four honest-majority checkpoint committees validate the intermediate state and transfers the verified state; each committee acts exactly once, using 3 threshold decryptions.

The last computation committee first broadcasts encryptions of its output-wire shares and tag shares with NIZKs of plaintext knowledge, and only then commits to and opens the seed shares determining the challenge $\mathbf{r}'$. Fixing the ciphertexts before the challenge exists is what prevents a corrupt committee from choosing contributions that cancel under a known $\mathbf{r}'$. Committee $\mathcal{C}_{i,1}^{\mathsf{Check}}$ verifies the NIZKs and homomorphically aggregates the encrypted shares into the compact candidate state $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}$, while the previous checkpoint is retained until the $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}$ is checked. However, verifying the MAC equation involves a product of two secrets, the key $\alpha$ and a linear combination of private wire values, which LHE cannot multiply. Writing $Y = \sum_k r'_k y_k$, $\Gamma = \sum_k r'_k \gamma(y_k)$ and $S = \sum_j \big(\sum_l \gamma_l^j - \alpha o^j\big)$, and using a fresh authenticated value $(R, \alpha R)$ from preprocessing, it forms encryptions of $Y + R$ and of $\sigma = S - (\Gamma + \alpha R)$ by linear homomorphism alone and threshold-decrypts $Y + R$ to $\mathcal{C}_{i,2}^{\mathsf{Check}}$. Since

$R$ is a fresh uniform value, this reveals nothing about the state and makes the remaining computation linear.

Committee $\mathcal{C}_{i,2}^{\mathsf{Check}}$ uses one scalar multiplication and one addition to compute $\mathsf{ct}_c := (y' \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{ct}_\sigma$, encrypting $c = \alpha(Y + R) + \sigma = S + \alpha Y - \Gamma$ (such that the MAC is valid if $c = 0$). The value $c$ is never decrypted: doing so could reveal a linear function of $\alpha$, and $\alpha$ must remain usable after a failed attempt. Each party of $\mathcal{C}_{i,2}^{\mathsf{Check}}$ instead samples $r_l$ and broadcasts a rerandomized $r_l \boxdot \mathsf{ct}_c$ with an NIZK of correct scalar multiplication. Committee $\mathcal{C}_{i,3}^{\mathsf{Check}}$ sums the verified ciphertexts into an encryption of $r \cdot c$ for $r = \sum_l r_l$ and threshold-decrypts that. Since $r$ is uniform and independent of $c$, the released value is zero when $c = 0$ and uniform otherwise, revealing whether the era is valid and nothing further.

Committee $\mathcal{C}_{i,4}^{\mathsf{Check}}$ keeps the current state $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}$ if the released value is zero and every required message is present. Otherwise, it sets $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}$ to $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ (rolling back to the previous valid state) and helps re-execute the era with fresh committees in each pool. It creates fresh authenticated shares of the values in $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}$ for the first computation committee, as well as robust MAC key shares for the preprocessing committee, and distributes them via a single batch threshold decryption of $2K_i n_{\mathsf{Comp}} + n_{\mathsf{Prep}}$ ciphertexts. The MAC key $\alpha$ always remains encrypted and the checkpoint committee learns only whether the era state is valid.

**Verification Across Failed Attempts.** The decision to retry an era is public, so it must not become a selective-failure oracle on private wires. The check therefore covers the epoch openings and not only the encrypted intermediate state. Checking the output wires of the era's last sub-circuit alone would be insufficient: an error introduced into one input of a multiplication disappears from its output when the other input is zero, so an adversary could learn that other input from whether the era commits. Covering the masked openings used to evaluate that multiplication removes the branch, since an inconsistent change to a multiplicand causes an authentication failure in the corresponding opening whatever the other input was. With both covered, whether a checkpoint committee retains or rolls back an era's intermediate state is a deterministic function of whether the adversary deviated in a way that changes the committed state, and deviating undetected requires forging a MAC tag. Every retry uses fresh shares of the latest valid state and fresh preprocessing, so attempts are independent, and the security argument accounts for the whole sequence of attempts together with the final output. In the extension achieving GOD, we can invoke an MPC protocol with GOD taking fresh shares of the latest valid state as input.

### 1.4 Further Related Works

We now discuss other relevant related works that are not already discussed in the remainder of the introduction.

**Specific Purpose Protocols.** Specific-purpose protocols in the YOSO MPC setting have been proposed for the tasks of threshold cryptography [10, 37] and random beacons [51, 52] (with worst-case corruption).

**Communication Among (Anonymous) Dynamic Committees.** Communication towards anonymous randomly chosen parties as required by YOSO can be realized in different ways, including realizing [20,44,50] the role assignment mechanism from [43] or the notion of Encryption to the Future (ETF) [18]. Publicly verifiable secret sharing in the YOSO model was proposed in [21, 22]. However, sending messages to committees that will only be selected in the future (potentially including parties who are not yet online) requires an interactive protocol with auxiliary committees [18].

**Other Models.** Similar models that allows for dynamic committees with different particularities have been proposed in [1, 7, 27, 35, 36, 42, 47, 53].

## 2 Preliminaries

**Notation.** We denote the security parameter by $\kappa \in \mathbb{N}$. In threshold settings, we use $n$ to denote the number of parties and $t$ the corruption threshold such that $0 < t < n$. We use $[a, b]$ for $a, b \in \mathbb{Z}$,

$a \leq b$, to denote $\{a, a+1, \ldots, b-1, b\}$. $[b]$ denotes the set $[1, b]$. We denote the concatenation of $x$ and $y$ by $(x \| y)$. Given a set $\mathcal{X}$, we denote by $x \leftarrow\!\$\ \mathcal{X}$ the sampling of a value $x$ from the uniform distribution on $\mathcal{X}$. A function $\mathsf{negl} : \mathbb{N} \to \mathbb{R}$ is negligible if it vanishes faster than any polynomial. We denote by $x = \mathsf{val}$ or $x \leftarrow \mathsf{val}$ the assignment of a value $\mathsf{val}$ to the variable $x$. We denote evaluating a $\mathsf{PPT}$ algorithm $\mathcal{S}$ that produces an output $\mathsf{out}$ from an input $\mathsf{in}$ with randomness $r \leftarrow\!\$\ \{0,1\}^*$ as $\mathsf{out} \leftarrow \mathcal{S}(\mathsf{in}; r)$, omitting the randomness when it is obvious or not explicitly required.

*Additive Secret Sharing.* Let $v \in \mathbb{F}$ and let $\mathbf{r} = (r_1, \ldots, r_{n-1}) \leftarrow\!\$\ \mathbb{F}^{n-1}$ be uniformly random. An additive sharing of $v$ among $n$ parties is computed as $\mathsf{AddShare}(v; \mathbf{r}) := ([v]_1, \ldots, [v]_n)$, where $[v]_i := r_i$ for $i \in [n-1]$ and $[v]_n := v - \sum_{i=1}^{n-1} r_i$. By construction, $\sum_{i=1}^{n} [v]_i = v$.

*SPDZ-Style Authenticated Sharing.* Let $\alpha \in \mathbb{F}$ be a global MAC key. An authenticated sharing of a value $v \in \mathbb{F}$ is defined as $\langle v \rangle := \big(\mathsf{AddShare}(v; \mathbf{r}),\ \mathsf{AddShare}(\alpha \cdot v; \mathbf{r}')\big)$, for independent randomness $\mathbf{r}, \mathbf{r}' \leftarrow\!\$\ \mathbb{F}^{n-1}$. Each party thus holds additive shares of both the value $v$ and its authentication tag $\alpha \cdot v$.

*Linearly Homomorphic Encryption Syntax.* We utilize a linearly homomorphic public-key encryption scheme $\mathsf{Lin\text{-}PKE} = (\mathsf{ParamsGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ as in Definition 13. To maintain concise protocol descriptions, we adopt the following operational syntax for homomorphic evaluations:

- **Addition:** $\mathsf{ct}_{m_1+m_2} := \mathsf{ct}_{m_1} \boxplus \mathsf{ct}_{m_2}$
- **Scalar Multiplication:** $\mathsf{ct}_{\alpha \cdot m_1} := \alpha \boxdot \mathsf{ct}_{m_1}$

*NIZK Proofs for Encryption.* To ensure parties provide valid ciphertexts, we use Zero-Knowledge Arguments of Knowledge as in Definition 14 for the relations $\mathsf{R}_{\mathsf{Enc}} := \{(\mathsf{pk}, \mathsf{ct}_x); (x, r) \mid \mathsf{ct}_x = \mathsf{Enc}(\mathsf{pk}, x; r)\}$ and $\mathsf{R}_{\mathsf{Scal}} := \{(\mathsf{pk}, \mathsf{ct}, \mathsf{ct}'); (\mu, s) \mid \mathsf{ct}' = (\mu \boxdot \mathsf{ct}) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; s)\}$, together with their conjunction over a shared witness,

$$\mathsf{R}_{\mathsf{Auth}} := \{\, (\mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}, \mathsf{ct}'); (\mu, r, s) \;\big|\; \mathsf{ct} = \mathsf{Enc}(\mathsf{pk}, \mu; r) \wedge \mathsf{ct}' = (\mu \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; s)\},$$

which certifies that $\mathsf{ct}$ and $\mathsf{ct}'$ encrypt $\mu$ and $\alpha \cdot \mu$ for the *same* $\mu$. We denote non-interactive proof generation as $\pi \leftarrow \mathsf{Prove}(x, w)$ and verification as $\mathsf{Verify}(x, \pi) \in \{0, 1\}$, writing $\pi_v^{\mathsf{Enc}}$, $\pi_v^{\mathsf{Scal}}$ and $\pi_v^{\mathsf{Auth}}$ for arguments for $\mathsf{R}_{\mathsf{Enc}}$, $\mathsf{R}_{\mathsf{Scal}}$ and $\mathsf{R}_{\mathsf{Auth}}$ respectively, subscripted by the value $v$ they concern. In each case the statement is the tuple to the left of the semicolon and the witness the tuple to its right.

**Universal Composability.** We work in the Universal Composability framework of Canetti [19]. Protocols proven secure in this framework are guaranteed to retain their security guarantees even under arbitrary concurrent composition. Intuitively, UC security is established by showing that for every adversary there exists a simulator such that an environment cannot distinguish a real world execution of the protocol with the adversary from an ideal world execution of an ideal functionality with the simulator. When this condition is satisfied, it is said that the protocol UC-realizes the ideal functionality. We refer the reader to [19] for the full definition.

**Threshold Encryption Functionality.** The functionality $\mathcal{F}_{\mathsf{TE}}$ (Figure 1) abstracts a standard threshold encryption scheme, providing key generation and threshold decryption for a fixed public-key encryption primitive. The functionality is a variant of the threshold encryption functionality of [17], adapted to our committee-based execution model. In our construction, $\mathcal{F}_{\mathsf{TE}}$ serves as the trusted primitive for state persistence. Checkpoint committees invoke this functionality to jointly decrypt MAC key shares and verification results, as well as facilitating the secure transfer of authenticated shares across era boundaries. In order to perform these operations, we additionally require the threshold encryption scheme to be linearly homomorphic. By modelling this component as an ideal functionality, the higher-level protocol remains agnostic to the specific threshold encryption scheme used to realize the primitive. Besides the protocol of [17], this functionality can also be realized based on the Paillier encryption scheme [54] as shown in [26] and by schemes based on class groups [16,23].

**Public Parameters:** Committees $\{\mathcal{C}_1, \cdots, \mathcal{C}_m\}$, Initialising Committees $\mathcal{C}_{\mathsf{init1}}$, $\mathcal{C}_{\mathsf{init2}}$.
**Init:** Upon receiving $(\mathsf{init}, sid, 1^\kappa, q)$ from all parties in $\mathcal{C}_{\mathsf{init}}$, $\mathcal{F}_{\mathsf{TE}}$ generates $\mathsf{pp}_{\mathsf{TE}} \leftarrow \mathsf{ParamsGen}(1^\kappa, q)$. It stores $\mathsf{pp}_{\mathsf{TE}}$ and sends them to all parties in $\mathcal{C}_1, \cdots, \mathcal{C}_m$. $\mathcal{F}_{\mathsf{TE}}$ ensures that this method is called only once, and before all other calls.
**KeyGen:** Upon receiving $(\mathsf{KGen}, sid)$ from all parties in $\mathcal{C}_{\mathsf{init2}}$, $\mathcal{F}_{\mathsf{TE}}$ obtains $(\mathsf{pk}_{\mathsf{TE}}, \mathsf{sk}_{\mathsf{TE}}) \leftarrow \mathsf{KGen}(\mathsf{pp}_{\mathsf{TE}})$. It outputs $\mathsf{pk}_{\mathsf{TE}}$ to all parties in $\mathcal{C}_1, \cdots, \mathcal{C}_m$ and stores $\mathsf{sk}_{\mathsf{TE}}$. It ensures that this method is called only once and before any call to $\mathsf{Dec}$.
**Decrypt:** Upon receiving $(\mathsf{Dec}, sid, \mathbf{ct}, \mathcal{P}_{\mathcal{R}})$ from all honest parties in $\mathcal{C}_i$ where $\mathbf{ct}$ is a set of $\ell$ ciphertexts and $\mathcal{P}_{\mathcal{R}}$ is a set of $\ell$ sets of receiver parties $\mathcal{P}_{\mathcal{R}}[k]$ for $k = 1, \ldots, \ell$, $\mathcal{F}_{\mathsf{TE}}$ computes $m_k := \mathsf{Dec}(\mathsf{sk}_{\mathsf{TE}}, \mathsf{ct}_k)$ for all $\mathsf{ct}_k \in \mathbf{ct}$, and outputs $m_k$ to all parties in $\mathcal{P}_{\mathcal{R}}[k]$. Ignore all future messages from parties in $\mathcal{C}_i$.

**Figure 1:** Functionality $\mathcal{F}_{\mathsf{TE}}$

## 3 Secure Computation – Model and Construction

We present ideal functionality $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ for Eras MPC evaluating a layered arithmetic circuit $\mathbf{C}$ over a prime field $\mathbb{F}_p$ and Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$, which UC-realizes $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$. The protocol employs three kinds of committees: preprocessing committees supply correlated randomness, computation committees evaluate sub-circuits gate-by-gate, and checkpoint committees validate the computation state at era boundaries. The central design choice is to defer MAC-tag verification to era boundaries, allowing computation committees to tolerate a fully dishonest majority while honest-majority work is amortized to infrequent checkpoint steps.

### 3.1 Eras MPC model and Ideal Functionality $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$

We define an ideal functionality $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ (Figure 2) capturing the security goal of Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$: the UC-secure evaluation of a layered arithmetic circuit $\mathbf{C}$ over $\mathbb{F}_p$, structured for *optimistic, era-based execution*. The circuit is partitioned into $l \cdot m$ bounded-depth sub-circuits $\{\mathbf{C}_{i,j}\}_{i\in[l],j\in[m]}$, each of constant depth $\delta$; we group $m$ consecutive sub-circuits into an *era*, the unit at which progress is committed or rolled back atomically. Sub-circuits are evaluated in sequence by computation committees drawn one at a time from an ordered pool $\mathcal{P}_{\mathsf{Comp}}$; on a detected deviation, the current era is retried from its start with the next committee in the pool. The committee structure, roles, and deferred-verification mechanism needed to realize this evaluation when the computation committee may be fully dishonest are introduced in Section 3.1 below.

The functionality maintains a circuit pointer $(i, j)$ and a counter $\mathsf{ctr}$ recording how many computation committees have been consumed from $\mathcal{P}_{\mathsf{Comp}}$, and models era failure by an explicit $\mathsf{EraAbort}$ from the adversary, capturing a computation committee that goes silent or deviates anywhere within the era. The resulting semantics are given in Figure 2; what matters for the construction is that each committed era becomes a stable recovery point, since a retry rolls back only to the start of the current era and never past it, and that the functionality aborts outright once too few committees remain in $\mathcal{P}_{\mathsf{Comp}}$ to finish.

Tolerating a fully dishonest majority within each era's computation committee requires deferring correctness verification to periodic checkpoints — which in turn requires preprocessing and checkpoint committees beyond the computation committees used above. We introduce this additional committee structure next.

**Roles and Committees.** The protocol employs three kinds of committees, each associated with a distinct type of role. A *role assignment* determines which parties form each committee and is treated as a parameter of the instantiation. The three committee kinds, their roles, and required corruption bounds are:

- **Preprocessing committees** ($\mathcal{R}_i^{\mathsf{Prep}}$, $n_{\mathsf{Prep}}$ parties, $t_{\mathsf{Prep}} < n_{\mathsf{Prep}}/2$): Parties in preprocessing roles for era $i$ generate all input-independent correlated randomness: authenticated Beaver triples, intra-era masking values, and masked-verification pairs for the era-boundary MAC check. An honest majority ensures the MAC key $\alpha$ is correctly distributed and the triples are well-formed.
- **Computation committees** ($\mathcal{R}_{i,j}^{\mathsf{Comp}}$, $n_{\mathsf{Comp}}$ parties, $t_{\mathsf{Comp}} < n_{\mathsf{Comp}}$): Parties in computation roles for sub-circuit $\mathbf{C}_{i,j}$ evaluate it gate-by-gate, consuming the correlated randomness for era $i$.

**Public Parameters:** A input committee $\mathcal{C}_{\mathsf{in}}$, an output committee $\mathcal{C}_{\mathsf{out}}$, and an ordered pool (*i.e.* a set) of computation committees $\mathcal{P}_{\mathsf{Comp}} = \{\mathcal{C}_1^{\mathsf{Comp}}, \ldots, \mathcal{C}_{N_{\mathsf{Comp}}}^{\mathsf{Comp}}\}$.

**Initialize:** On input $(sid, \mathsf{init}, p, \mathbf{C} = \{\mathbf{C}_{i,j}\}_{i\in[l],j\in[m]})$ from all $P_k \in \mathcal{C}_{\mathsf{in}}$, store $p$ and $\mathbf{C}$, initialize a circuit pointer $(i, j) \leftarrow (1, 1)$, and a used committee counter $\mathsf{ctr} \leftarrow 1$.

**Input:** On input $(sid, \mathsf{input}, P_k, x_k)$ from $P_k \in \mathcal{C}_{\mathsf{in}}$ and $(sid, \mathsf{input}, P_k)$ from all other parties in $\mathcal{C}_{\mathsf{in}}$, store $x_k$. Once all parties in $\mathcal{C}_{\mathsf{in}}$ have sent their inputs, select $\{\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{ctr}+m-1}^{\mathsf{Comp}}\} \in \mathcal{P}_{\mathsf{Comp}}$ as the active computation committees for Era 1; and send $(sid, \mathsf{EraSuccess}, 0)$ to $\{\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{ctr}+m-1}^{\mathsf{Comp}}\}$.

**Evaluate:** Upon receiving $(sid, \mathsf{Eval}, i', j')$ from all parties in $\mathcal{C}_{\mathsf{ctr}+j'}^{\mathsf{Comp}}$, ignore if $(i, j) \neq (i', j')$; otherwise do the following:

1. Store the vector $\mathbf{y}_{i,j}$ of outputs of $\mathbf{C}_{i,j}$ evaluated on:
   - $\{x_k\}$ if $i = 1$ and $j = 1$; // original inputs
   - the outputs of $\mathbf{C}_{i,j-1}$ if $1 < j \leq m$; // previous circuit outputs in same Era
   - the outputs of $\mathbf{C}_{i-1,m}$ if $j = 1$ and $1 < i \leq l$; // previous era outputs
2. If $j < m$: set $(i, j) \leftarrow (i, j + 1)$; // evaluate the next circuit in Era $i$
3. If $j = m$: // the computation of circuits in Era $i$ is completed
   - If $i = l$, send $(sid, \mathsf{OutputSuccess})$ to $\mathcal{C}_{\mathsf{out}}$. // last Era output is ready
   - Else set $\mathsf{ctr} \leftarrow \mathsf{ctr} + m$; // $m$ computation committees used in the next Era
   - If $(l - i) \cdot m > N_{\mathsf{Comp}} - \mathsf{ctr}$, output $\perp$ to all parties in $\mathcal{C}_{\mathsf{out}}$ and halt. // abort if not enough committees left in $\mathcal{P}_{\mathsf{Comp}}$ to finish the computation
   - If $i < l$: set $(i, j) \leftarrow (i + 1, 1)$; select $\{\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{ctr}+m-1}^{\mathsf{Comp}}\} \in \mathcal{P}_{\mathsf{Comp}}$ as the active computation committees for Era $i$; send $(sid, \mathsf{EraSuccess}, i - 1)$ to $\{\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{ctr}+m-1}^{\mathsf{Comp}}\}$. // proceed to next Era

**Era Abort:** On input $(sid, \mathsf{EraAbort})$ from the adversary $\mathcal{S}$, set $\mathsf{ctr} \leftarrow \mathsf{ctr} + m$ and $(i, j) \leftarrow (i, 1)$. If $(l - i) \cdot m > N_{\mathsf{Comp}} - \mathsf{ctr}$, output $\perp$ to all parties in $\mathcal{C}_{\mathsf{out}}$ and halt. Otherwise, select $\{\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{ctr}+m-1}^{\mathsf{Comp}}\} \in \mathcal{P}_{\mathsf{Comp}}$ as the active computation committees for Era $i$; send $(sid, \mathsf{EraFail}, i)$ to $\{\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{ctr}+m-1}^{\mathsf{Comp}}\}$.

**Output:** On input $(sid, \mathsf{output}, P_k)$ from all honest parties in $\mathcal{C}_{\mathsf{out}}$, if $(sid, \mathsf{OutputSuccess})$ has been sent, send $(sid, \mathsf{Output}, \mathbf{y}_{l,m})$ to all parties in $\mathcal{C}_{\mathsf{out}}$; otherwise output $\perp$.

**Figure 2:** Ideal Functionality $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ for Eras MPC (// comments in red)

Security holds even when all but one party are corrupted, since integrity is enforced only at the era boundary.

– **Checkpoint committees** ($\mathcal{R}_{i,1}^{\mathsf{Check}}$–$\mathcal{R}_{i,4}^{\mathsf{Check}}$, $n_{\mathsf{Check}}$ parties each, $t_{\mathsf{Check}} < n_{\mathsf{Check}}/2$): Four successive committees filling checkpoint roles at the close of era $i$ validate the accumulated MAC transcript, commit or roll back the era, hand off the verified state to the next computation committee, and distribute fresh preprocessing key material.

**Roles within an Era.** Each sub-circuit $\mathbf{C}_{i,j}$ is assigned to computation roles $\mathcal{R}_{i,j}^{\mathsf{Comp}}$; era $i$ also preprocessing roles $\mathcal{R}_i^{\mathsf{Prep}}$ and checkpoint roles $\mathcal{R}_{i,1}^{\mathsf{Check}}$–$\mathcal{R}_{i,4}^{\mathsf{Check}}$, respectively assigned to committees in the pools introduced above. The era remains the unit of atomic progress: either all $m$ sub-circuits of era $i$ successfully verify in the checkpoint phase, or the entire era is retried with fresh committees.

**Optimistic Execution.** Within each era, computation committees evaluate sub-circuits gate-by-gate under SPDZ, accumulating an unauthenticated transcript; MAC-tag verification is *deferred to era boundary*, where checkpoint committees validate the full transcript in one pass and either commit the era or retry it with a fresh assignment. This is what separates the dishonest-majority computation path from honest-majority validation step, as explained in Section 1.

**Role Assignment and Committee Pools.** The protocol is stated in terms of abstract committee roles; role assignment is a parameter of the instantiation. In our primary setting, all committees are fixed before execution begins, as in Fluid MPC [25] and Multi-Layered MPC [33]. Role assignment for more dynamic settings, including dynamic participation, anonymous selection, and delayed joining, is addressed by a rich body of existing work [18, 20, 43, 44, 50]; any such mechanism yielding committees satisfying the required corruption threshold bounds can instantiate our construction. Anonymous YOSO-style assignment, in which parties remain unidentified until they speak, additionally requires receiver-anonymous channels, which are outside the scope of this work.

Differently from YOSO, Fluid MPC and Layered MPC where each role is assigned to exactly one party, we consider that parties may fail to execute their role and that the role will be reassigned to a new party. Moreover, we do not require each role to execute a single round, allowing each role to execute different constant numbers of rounds. In particular, we consider that a set of roles may be assigned to a certain committee who then fails to execute their roles after a certain number of rounds, causing the roles to be reassigned to a committee who has not yet acted. Moreover, protocols may employ multiple active committees in parallel, each with different corruption thresholds and acting for a different number of rounds. We assume that there exist publicly ordered committee pools containing fresh committees with prescribed corruption thresholds to which roles may be (re)assigned. If a given committee fails in executing the protocol steps of a certain set of roles, the protocol may require the next committee in a pool to be assigned to those roles in order re-try executing the required steps.

Realizing $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ with dishonest-majority computation committees requires preprocessing committees to supply authenticated correlated randomness before each era begins. We formalize this output as an ideal functionality.

### 3.2 Preprocessing Functionality $\mathcal{F}_{\mathsf{Prep}}$

Functionality $\mathcal{F}_{\mathsf{Prep}}$ (Figure 3) captures the output of preprocessing committees: the input-independent correlated randomness that computation committees need to evaluate sub-circuits. It has two modes: one that generates masked input representations before execution begins, and one that generates triples and other correlations needed for each era.

**PrepInput** generates masked representations of the input layer. For each input party $P_j \in \mathcal{C}_0$, it samples a random mask $r_j$, computes its MAC tag $\gamma(r_j) = \alpha \cdot r_j$, and outputs ciphertexts $\mathsf{ct}_{r_j}, \mathsf{ct}_{\gamma(r_j)}$. These allow the first computation committee to obtain correctly authenticated SPDZ shares of input wires without any committee learning $\alpha$ in the clear.

**PrepCompute** generates the correlated material for era $i$: authenticated Beaver triples $(\langle a_{j,k}\rangle, \langle b_{j,k}\rangle, \langle c_{j,k}\rangle)$ with $c_{j,k} = a_{j,k} \cdot b_{j,k}$ (one per multiplication gate in $\mathbf{C}_{i,j}$); independent intra-era masking values $[\beta_k^{i,j}]^j, [\eta_k^{i,j}]^j$ and $[\beta_k^{i,j}]^{j+1}, [\eta_k^{i,j}]^{j+1}$, masking the wire value and its tag respectively, enabling threshold-decryption-free state handoffs between consecutive committees; and a masked-verification pair $(\mathsf{ct}_R, \mathsf{ct}_{\gamma(R)})$ for blinding the era-boundary MAC check. We assume $\mathcal{F}_{\mathsf{Prep}}$ is instantiated in an honest-majority setting; a concrete realization is given in Section 3.4.

**Public Parameters:** An ordered pool (*i.e.* a set) of computation committees $\mathcal{P}_{\mathsf{Comp}} = \{\mathcal{C}_1^{\mathsf{Comp}}, \ldots, \mathcal{C}_{N_{\mathsf{Comp}}}^{\mathsf{Comp}}\}$ and checkpoint committees $\mathcal{P}_{\mathsf{Check}} = \{\mathcal{C}_1^{\mathsf{Check}}, \ldots, \mathcal{C}_{N_{\mathsf{Check}}}^{\mathsf{Check}}\}$.

**PrepInput:** Upon receiving $(\mathsf{PrepInput}, sid, \mathcal{C}_0, (\mathsf{pk}, \alpha_i))$ from all honest parties in $\mathcal{C}_0^{\mathsf{Prep}} \setminus \mathcal{I}_0^{\mathsf{Prep}}$, $\mathcal{F}_{\mathsf{Prep}}$ does the following:

1. Compute $\alpha = \mathsf{Rec}(\{\alpha_i\}_i)$. Send $\mathsf{pk}$ to $\mathcal{S}$.
2. Sample uniformly random values $\{r_i\}_{i\in[n_{\mathsf{in}}]}$ from $\mathbb{F}_p$. Compute $\gamma(r_i) := \alpha \cdot r_i$.
3. For each $i \in [n_{\mathsf{in}}]$, compute $\mathsf{ct}_{r_i} \leftarrow \mathsf{Enc}(\mathsf{pk}, r_i), \mathsf{ct}_{\gamma(r_i)} \leftarrow \mathsf{Enc}(\mathsf{pk}, \gamma(r_i))$.
4. Broadcast $\{\mathsf{ct}_{r_i}, \mathsf{ct}_{\gamma(r_i)}\}_{i\in[n_{\mathsf{in}}]}$ to $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$

**PrepCompute:** Upon receiving $(\mathsf{PrepCompute}, sid, i, \mathsf{CompCtr}, \mathsf{CheckCtr}, \{\mathsf{int}_{i,j}\}_{j\in[m]}, (\mathsf{pk}, \alpha_l))$ from all honest $P_l \in \mathcal{C}_{\mathsf{PrepCtr}}^{\mathsf{Prep}} \setminus \mathcal{I}_{\mathsf{PrepCtr}}^{\mathsf{Prep}}$, $\mathcal{F}_{\mathsf{Prep}}$ does the following:

1. Compute $\alpha = \mathsf{Rec}(\{\alpha_l\}_l)$.
2. Sample a uniformly random value $R$ from $\mathbb{F}_p$. Compute $\gamma(R) = \alpha \cdot R$
3. Compute $\mathsf{ct}_R \leftarrow \mathsf{Enc}(\mathsf{pk}, R), \mathsf{ct}_{\gamma(R)} \leftarrow \mathsf{Enc}(\mathsf{pk}, \gamma(R))$.
4. For each $j \in [m], k \in [\mathsf{int}_{i,j}]$, sample uniformly $a_{j,k}, b_{j,k}, \beta_k^{i,j}, \eta_k^{i,j}$ from $\mathbb{F}_p$, and compute $c_{j,k} = a_{j,k} \cdot b_{j,k}$. The handoff masks $\beta_k^{i,j}$ and $\eta_k^{i,j}$ are sampled independently of one another.
5. Computes SPDZ-sharings $\langle a_{j,k}\rangle, \langle b_{j,k}\rangle, \langle c_{j,k}\rangle$ and, once per computation committee, $\langle 1\rangle^j$, all wrt $\alpha$, and plain additive sharings $[\beta_k^{i,j}]^j, [\eta_k^{i,j}]^j, [\beta_k^{i,j}]^{j+1}, [\eta_k^{i,j}]^{j+1}$, where the sharings indexed $j$ and $j+1$ of the same value are independent.
6. For $k \in [\mathsf{int}_{i,j}]$, output shares of $\langle a_{j,k}\rangle, \langle b_{j,k}\rangle, \langle c_{j,k}\rangle, \langle 1\rangle^j, [\beta_k^{i,j}]^j, [\eta_k^{i,j}]^j$ to $\mathcal{C}_{\mathsf{CompCtr}+j-1}^{\mathsf{Comp}}$; if $j < m$, shares of $[\beta_k^{i,j}]^{j+1}, [\eta_k^{i,j}]^{j+1}$ to $\mathcal{C}_{\mathsf{CompCtr}+j}^{\mathsf{Comp}}$; and broadcast $\mathsf{ct}_R, \mathsf{ct}_{\gamma(R)}$ to $\mathcal{C}_{\mathsf{CheckCtr}}^{\mathsf{Check}}$.

**Figure 3:** Functionality $\mathcal{F}_{\mathsf{Prep}}$

Distributing the MAC key $\alpha$ to preprocessing committees and generating per-era randomness requires jointly producing encryptions of fresh random field elements without any party learning the

plaintext. We use a standard two-committee coin toss $\Pi_{\mathsf{CoinToss}}$ for this purpose: every party of the first committee broadcasts an encryption of a uniformly random $r_i \in \mathbb{F}_p$ together with a NIZK proof of correct encryption, and the second committee discards malformed contributions and homomorphically aggregates the rest into $\mathsf{ct}_r := \boxplus_i \mathsf{ct}_{r_i}$, encrypting $r = \sum_i r_i$, which remains uniform and unknown to every party as long as one contributor is honest. It is invoked during setup to produce $\mathsf{ct}_\alpha$ and the MAC key polynomial coefficients, and at each era boundary to produce fresh preprocessing key material and the $2K(n_{\mathsf{Comp}} - 1)$ per-wire masking ciphertexts used for state expansion. The protocol is given in full in Appendix A.4.

Finally, $\mathcal{F}_{\mathsf{TE}}$ (Figure 1) is a threshold linearly homomorphic encryption functionality. It is used to distribute MAC key $\alpha$ to preprocessing committees via threshold decryption, and to carry authenticated computation state across era boundaries as a compact ciphertext that no single committee can decrypt alone.

### 3.3 Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$

With the model and all supporting functionalities in place, we present $\Pi_{\mathsf{Era\text{-}MPC}}$ in full. The protocol realizes $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ by executing the optimistic pipeline of Section 3.1 through three interleaved phases:

- **Computation.** Within era $i$, computation committees $\mathcal{R}^{\mathsf{Comp}}_{i,1}, \ldots, \mathcal{R}^{\mathsf{Comp}}_{i,m}$ evaluate sub-circuits $\mathbf{C}_{i,1}, \ldots, \mathbf{C}_{i,m}$ gate-by-gate under SPDZ, consuming one preprocessed triple per multiplication.
- **Intra-era state handoff.** At the boundary between $\mathbf{C}_{i,j}$ and $\mathbf{C}_{i,j+1}$, committee $\mathcal{R}^{\mathsf{Comp}}_{i,j}$ passes its output wires to $\mathcal{R}^{\mathsf{Comp}}_{i,j+1}$ under the independent preprocessed masks $\beta^{i,j}_k$ and $\eta^{i,j}_k$, with no threshold decryption and a single broadcast per output wire per party.
- **Era-boundary consistency check and state transfer.** At the close of era $i$ (after evaluating $\mathbf{C}_{i,m}$), a relay of four checkpoint committees $\mathcal{R}^{\mathsf{Check}}_{i,1}, \ldots, \mathcal{R}^{\mathsf{Check}}_{i,4}$ validates the accumulated MAC transcript, decides whether to commit or roll back, and initialises the next era. This is the only global synchronisation point in the protocol; the entire computation path between two era boundaries involves no checkpoint committee interaction.

Since computation committees may be fully corrupted, the era transcript cannot be trusted until it passes the consistency check. Deferring verification to the era boundary is what makes dishonest-majority computation feasible: a corrupt committee cannot forge a state that satisfies the MAC equations without knowing $\alpha$, and this is enforced by the checkpoint committees at low amortised cost. Figure 4 specifies the setup and input phases; Figure 5 details the computation and intra-era handoffs; Figures 6, 7 specify the era-boundary consistency check and state transfer.

**Committee Pool Organization.** The protocol maintains three kinds of publicly ordered pools as a reserve for fault recovery, each with a dedicated counter and role assignment:

- **Computation pool** $\mathcal{P}_{\mathsf{Comp}} := \{\mathcal{C}^{\mathsf{Comp}}_1, \cdots, \mathcal{C}^{\mathsf{Comp}}_{N_{\mathsf{Comp}}}\}$: Contains $N_{\mathsf{Comp}} \gg l \cdot m$ committees, each with $n_{\mathsf{Comp}}$ parties. Counter $\mathsf{CompCtr}$ counts how many computation committees have been consumed from the pool. For era $i$, roles $\{\mathcal{R}^{\mathsf{Comp}}_{i,j}\}_{j\in[m]}$ are instantiated by the contiguous block $\{\mathcal{C}^{\mathsf{Comp}}_{\mathsf{CompCtr}}, \ldots, \mathcal{C}^{\mathsf{Comp}}_{\mathsf{CompCtr}+m-1}\}$, matching the counter $\mathsf{ctr}$ of $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ exactly. Upon rollback or commit, $\mathsf{CompCtr} \leftarrow \mathsf{CompCtr} + m$ because each era uses $m$ computation committees.
- **Preprocessing pool** $\mathcal{P}_{\mathsf{Prep}} := \{\mathcal{C}^{\mathsf{Prep}}_1, \cdots, \mathcal{C}^{\mathsf{Prep}}_{N_{\mathsf{Prep}}}\}$: Contains $N_{\mathsf{Prep}} \gg l$ committees, each of size $n_{\mathsf{Prep}}$. Counter $\mathsf{PrepCtr}$ counts how many preprocessing committees have been consumed from the pool. For era $i$, role $\mathcal{R}^{\mathsf{Prep}}_i$ is instantiated by the single committee $\mathcal{C}^{\mathsf{Prep}}_{\mathsf{PrepCtr}}$. Upon rollback or commit, $\mathsf{PrepCtr} \leftarrow \mathsf{PrepCtr} + 1$ because each era uses exactly one preprocessing committee.
- **Checkpoint pool** $\mathcal{P}_{\mathsf{Check}} := \{\mathcal{C}^{\mathsf{Check}}_1, \cdots, \mathcal{C}^{\mathsf{Check}}_{N_{\mathsf{Check}}}\}$: Contains $N_{\mathsf{Check}} \gg l \cdot 4$ committees, each of size $n_{\mathsf{Check}}$. Counter $\mathsf{CheckCtr}$ counts how many checkpoint committees have been consumed from the pool. For era $i$, roles $\{\mathcal{R}^{\mathsf{Check}}_{i,k}\}_{k\in\{1,2,3,4\}}$ are instantiated by the contiguous block $\{\mathcal{C}^{\mathsf{Check}}_{\mathsf{CheckCtr}}, \mathcal{C}^{\mathsf{Check}}_{\mathsf{CheckCtr}+1}, \mathcal{C}^{\mathsf{Check}}_{\mathsf{CheckCtr}+2}, \mathcal{C}^{\mathsf{Check}}_{\mathsf{CheckCtr}+3}\}$. Upon rollback or commit, $\mathsf{CheckCtr} \leftarrow \mathsf{CheckCtr} + 4$ because each era uses exactly four checkpoint committees.

Each committee in a pool is consumed at most once. Upon rollback, the next block from each pool is assigned to retry era $i$; upon commit, the next block from each pool is assigned to era $i + 1$.

**Public Parameters:** Committee pools $\mathcal{P}_{\mathsf{Comp}} := \{\mathcal{C}_1^{\mathsf{Comp}}, \cdots, \mathcal{C}_{N_{\mathsf{Comp}}}^{\mathsf{Comp}}\}, \mathcal{P}_{\mathsf{Prep}} := \{\mathcal{C}_1^{\mathsf{Prep}}, \cdots, \mathcal{C}_{N_{\mathsf{Prep}}}^{\mathsf{Prep}}\}, \mathcal{P}_{\mathsf{Check}} := \{\mathcal{C}_1^{\mathsf{Check}}, \cdots, \mathcal{C}_{N_{\mathsf{Check}}}^{\mathsf{Check}}\}$ as described above; circuit $\mathbf{C}$ partitioned into sub-circuits $\{\mathbf{C}_{i,j}\}_{i\in[l],j\in[m]}$; initializing committees $\mathcal{C}_{\mathsf{init0}}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{init1}}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{init2}}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{init3}}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$, used once to initialize the parameters of the protocol; input committee $\mathcal{C}_{\mathsf{in}} := \mathcal{C}_0$ and output committee $\mathcal{C}_{\mathsf{out}} := \mathcal{C}_1$.

**State Initialization:** Maintain three pool counters $\mathsf{CompCtr} := 1, \mathsf{PrepCtr} := 1$, and $\mathsf{CheckCtr} := 1$. Initialize an era pointer $(i, j) := (1, 1)$, to track the current era $i$ and the sub-circuit $\mathbf{C}_{i,j}$ within it. Select $\{\mathcal{R}_{1,j}^{\mathsf{Comp}}\}_{j\in[m]} \leftarrow \{\mathcal{C}_{\mathsf{CompCtr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{CompCtr}+m-1}^{\mathsf{Comp}}\}$, $\mathcal{R}_1^{\mathsf{Prep}} \leftarrow \mathcal{C}_{\mathsf{PrepCtr}}^{\mathsf{Prep}}$, and $\{\mathcal{R}_{1,k}^{\mathsf{Check}}\}_{k\in[4]} \leftarrow \{\mathcal{C}_{\mathsf{CheckCtr}}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{CheckCtr}+1}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{CheckCtr}+2}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{CheckCtr}+3}^{\mathsf{Check}}\}$.

**Setup Phase:**

1. Parties in $\mathcal{C}_{\mathsf{init0}}^{\mathsf{Check}}$ send $(\mathsf{init}, sid, 1^\kappa)$ with $\mathcal{F}_{\mathsf{TE}}$; parties in $\mathcal{C}_{\mathsf{init1}}^{\mathsf{Check}}$ receive $\mathsf{pp}_{\mathsf{TE}}$.
2. Parties in $\mathcal{C}_{\mathsf{init1}}^{\mathsf{Check}}$ send $(\mathsf{KGen}, sid)$ with $\mathcal{F}_{\mathsf{TE}}$; parties in $\mathcal{C}_{\mathsf{init2}}^{\mathsf{Check}}$ receive $\mathsf{pk}_{\mathsf{TE}}$.
3. Each party $P_i \in \mathcal{C}_{\mathsf{init2}}^{\mathsf{Check}}$ run $t_{\mathsf{Prep}} + 1$ instantiations $\Pi_{\mathsf{CoinToss}}$ to output to all parties in $\mathcal{C}_{\mathsf{init3}}^{\mathsf{Check}}$ the values $\mathsf{ct}_\alpha, \mathsf{ct}_{c_1}, \cdots, \mathsf{ct}_{c_t}$.
4. Each party $P_i \in \mathcal{C}_{\mathsf{init3}}^{\mathsf{Check}}$ having received $\mathsf{ct}_\alpha, \mathsf{ct}_{c_1}, \cdots, \mathsf{ct}_{c_t}$ as part of $\Pi_{\mathsf{CoinToss}}$, computes $\mathsf{ct}_{\alpha_j}$ for $\alpha_j = f(j)$ where $f(x) = \alpha + c_1 x + c_2 x^2 + \cdots + c_t x^t$. Send $(\mathsf{Dec}, sid, \{\mathsf{ct}_{\alpha_1}, \cdots, \mathsf{ct}_{\alpha_{n_{\mathsf{Prep}}}}\}, \mathcal{C}_0^{\mathsf{Prep}})$ to $\mathcal{F}_{\mathsf{TE}}$, and output $\mathsf{ct}_\alpha$ to $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$.
5. Upon receiving $\alpha_k$ from $\mathcal{F}_{\mathsf{TE}}$, each $P_k \in \mathcal{C}_0^{\mathsf{Prep}}$ sends $(\mathsf{PrepInput}, sid, \mathcal{C}_0, (\mathsf{pk}, \alpha_k))$ to $\mathcal{F}_{\mathsf{Prep}}$. Parties in $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$ receive $\mathsf{ct}_{r_j}, \mathsf{ct}_{\gamma(r_j)}$ for all $P_j \in \mathcal{C}_0$ from $\mathcal{F}_{\mathsf{Prep}}$.
6. $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$ runs $2n_{\mathsf{in}} \cdot (n_{\mathsf{Comp}} - 1)$ instantiations of $\Pi_{\mathsf{CoinToss}}$ to output to all parties in $\mathcal{C}_{0,1}^{\mathsf{Check}}$ the values $\{(\mathsf{ct}_{r_{i,1}}, \mathsf{ct}_{\gamma(r_i)_1}), \cdots, (\mathsf{ct}_{r_{i,n_{\mathsf{Comp}}-1}}, \mathsf{ct}_{\gamma(r_i)_{n_{\mathsf{Comp}}-1}})\}_{i\in[n_{\mathsf{in}}]}$.
7. Parties in $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$ send $(\mathsf{Dec}, sid, \{\mathsf{ct}_{r_i}\}_{i\in[n_{\mathsf{in}}]}, \mathcal{P}_{\mathcal{R}})$ to $\mathcal{F}_{\mathsf{TE}}$, with $\mathcal{P}_{\mathcal{R}}$ addressing each $\mathsf{ct}_{r_i}$ to $\{P_i^0\}$, and broadcast $\{\mathsf{ct}_{r_i}, \mathsf{ct}_{\gamma(r_i)}\}_{i\in[n_{\mathsf{in}}]}$ and $\mathsf{ct}_\alpha$ to $\mathcal{C}_{0,1}^{\mathsf{Check}}$.

**Input Sharing Phase:**

1. For each party $P_i^0 \in \mathcal{C}_0$ with input $x_i$, receives $r_i$ from $\mathcal{F}_{\mathsf{TE}}$ and broadcasts $\epsilon_i = x_i - r_i$ to $\mathcal{C}_{0,1}^{\mathsf{Check}}$. If any party fails to broadcast, use a default input.
2. Each party in $\mathcal{C}_{0,1}^{\mathsf{Check}}$ does the following:
   (a) For $i \in [n_{\mathsf{in}}], j \in [n_{\mathsf{Comp}} - 1]$, $\mathsf{ct}_{[x_i]_j} := \mathsf{ct}_{r_{i,j}}, \mathsf{ct}_{[\gamma(x_i)_j]} := \mathsf{ct}_{\gamma(r_i)_j}$.
   (b) For $i \in [n_{\mathsf{in}}]$, set $\mathsf{ct}_{[x_i]_{n_{\mathsf{Comp}}}} := \mathsf{ct}_{\epsilon_i} \boxplus \mathsf{ct}_{r_i} \boxplus (-1 \boxdot \boxplus_{j=1}^{n_{\mathsf{Comp}}-1} \mathsf{ct}_{[x_i]_j})$.
   (c) For $i \in [n_{\mathsf{in}}]$, set $\mathsf{ct}_{[\gamma(x_i)]_{n_{\mathsf{Comp}}}} := (\epsilon_i \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{ct}_{\gamma(r_i)} \boxplus (-1 \boxdot \boxplus_{j=1}^{n_{\mathsf{Comp}}-1} \mathsf{ct}_{[\gamma(x_i)]_j})$.
   (d) Set and broadcast $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}} := \{\{\mathsf{ct}_{[x_i]_j}, \mathsf{ct}_{[\gamma(x_i)_j]}\}_{i\in[n_{\mathsf{in}}]}\}_{j\in n_{\mathsf{Comp}}}, \mathsf{ct}_\alpha$ to $\mathcal{C}_{0,2}^{\mathsf{Check}}$.
   (e) Execute $t_{\mathsf{Prep}}$ instantiations of $\Pi_{\mathsf{CoinToss}}$ such that $\mathcal{C}_{0,2}^{\mathsf{Check}}$ receive $\mathsf{ct}_{c_1}, \cdots, \mathsf{ct}_{c_t}$.
3. Having received $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}, \mathsf{ct}_\alpha, \mathsf{ct}_{c_1}, \cdots, \mathsf{ct}_{c_t}$, each party in $\mathcal{C}_{0,2}^{\mathsf{Check}}$ does the following:
   (a) Computes $\mathsf{ct}_{\alpha_j}$ for $\alpha_j = f(j)$ where $f(x) = \alpha + c_1 x + c_2 x^2 + \cdots + c_t x^t$.
   (b) Send $(\mathsf{Dec}, sid, (\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}, \{\mathsf{ct}_{\alpha_1}, \cdots, \mathsf{ct}_{\alpha_{n_{\mathsf{Prep}}}}\}), \mathcal{P}_{\mathcal{R}})$ to $\mathcal{F}_{\mathsf{TE}}$, and $\mathsf{ct}_{\alpha_j}$ to $\{P_j\} \subseteq \mathcal{R}_1^{\mathsf{Prep}}$. Broadcast $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}, \mathsf{ct}_\alpha$ to $\mathcal{R}_{1,1}^{\mathsf{Check}}$.
4. Each party $P_k \in \mathcal{R}_1^{\mathsf{Prep}}$ receives $\alpha_k$ from $\mathcal{F}_{\mathsf{TE}}$ and sends $(\mathsf{PrepCompute}, sid, i, \mathsf{CompCtr}, \mathsf{CheckCtr}, \{\mathsf{int}_{1,j}\}_{j\in[m]}, (\mathsf{pk}, \alpha_k))$ to $\mathcal{F}_{\mathsf{Prep}}$.

**Figure 4:** $\Pi_{\mathsf{Era\text{-}MPC}}$ — Setup and Input Phase

**Consistency Validation and State Transfer** The era-boundary relay (Figures 6, 7) executes as three sequential threshold decryptions across four committees: $\mathcal{R}_{i,1}^{\mathsf{Check}}$ aggregates the encrypted SPDZ shares from $\mathcal{R}_{i,m}^{\mathsf{Comp}}$, assembles $\mathsf{ct}_\sigma$ encoding the SPDZ2 MAC check, and decrypts $\mathsf{ct}_{y+R}$; $\mathcal{R}_{i,2}^{\mathsf{Check}}$ closes the MAC equation $\mathsf{ct}_c := (y' \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{ct}_\sigma$ encrypting $c = \sum_j (-\alpha \cdot o^j + \sum_l \gamma_l^j)$ and blinds it as $\mathsf{ct}_{c,l} := r_l \boxdot \mathsf{ct}_c$; $\mathcal{R}_{i,3}^{\mathsf{Check}}$ aggregates the blinded contributions and decrypts $\mathsf{ct}_{r\cdot c}$, so that what is released is $r \cdot c$ rather than $c$ itself; $\mathcal{R}_{i,4}^{\mathsf{Check}}$ commits if $r \cdot c = 0$ and rolls back to $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ otherwise, advancing $\mathsf{CompCtr}, \mathsf{PrepCtr}, \mathsf{CheckCtr}$ in either case. Each committed era serves as a stable recovery point; if any pool is exhausted, $\mathcal{F}_{\mathsf{TE}}$ can decrypt the last verified state for any fail-safe protocol providing guaranteed output delivery (Section 4).

**Theorem 1.** *For computation committees of $n_{\mathsf{Comp}}$ parties with $t_{\mathsf{Comp}}$ corruptions, preprocessing committees of $n_{\mathsf{Prep}}$ parties with $t_{\mathsf{Prep}}$ corruptions and checkpoint committees of $n_{\mathsf{Check}}$ parties with $t_{\mathsf{Check}}$ corruptions such that $n_{\mathsf{Comp}} \geq t_{\mathsf{Comp}} + 1$, $n_{\mathsf{Prep}} \geq 2t_{\mathsf{Prep}} + 1$ and $n_{\mathsf{Check}} \geq 2t_{\mathsf{Check}} + 1$, Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$ (Figures 4,5, 6, 7) UC-realizes $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ (Figure 2) in the $(\mathcal{F}_{\mathsf{Prep}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{TE}})$-hybrid model with security against a static adversary.*

**Computation Phase:**

1. For all $i \in [l]$ and $j \in [m]$, evaluate sub-circuit $\mathbf{C}_{i,j}$ by $\mathcal{R}^{\mathsf{Comp}}_{i,j}$:
   (a) Parties in $\mathcal{R}^{\mathsf{Comp}}_{i,j}$ receive SPDZ-sharings of the input wires to $\mathbf{C}_{i,j}$ from $\mathcal{F}_{\mathsf{TE}}$.
   (b) From $\mathcal{F}_{\mathsf{Prep}}$, for each multiplication gate, parties receive an authenticated multiplication triple $(\langle a\rangle, \langle b\rangle, \langle c\rangle, \langle 1\rangle)$. In addition, for each output wire $k$ of $\mathbf{C}_{i,j}$, the parties in $\mathcal{R}^{\mathsf{Comp}}_{i,j}$ and $\mathcal{R}^{\mathsf{Comp}}_{i,j+1}$ receive two *independent* sharings of the random values $\beta^{i,j}_k$ and $\eta^{i,j}_k$, denoted $[\beta^{i,j}_k]^j, [\eta^{i,j}_k]^j$ and $[\beta^{i,j}_k]^{j+1}, [\eta^{i,j}_k]^{j+1}$.
   (c) The parties in $\mathcal{R}^{\mathsf{Comp}}_{i,j}$ jointly evaluate $\mathbf{C}_{i,j}$ gate-by-gate as follows:
      – $\mathsf{add}(x, y)$: Compute $\langle x + y\rangle = \langle x\rangle + \langle y\rangle$ locally.
      – $\mathsf{multiply}(x, y)$: Use a multiplication triple $(\langle a\rangle, \langle b\rangle, \langle c\rangle)$. They open the values $\epsilon = x - a$ and $\rho = y - b$, and locally compute a sharing of the output $\langle z\rangle = \langle c\rangle + \epsilon \cdot \langle b\rangle + \rho \cdot \langle a\rangle + \epsilon \cdot \rho \cdot \langle 1\rangle$.
2. Verification and State Transition:
   (a) **CheckOpen:** Let $(o^j_1, \cdots, o^j_{t'})$ be the list of publicly opened values by Committee $\mathcal{R}^{\mathsf{Comp}}_{i,j}$. Let $\{[\gamma(o^j_1)], \cdots, [\gamma(o^j_{t'})]\}$ be the sharings of the set of tags for the opened values in Committee $\mathcal{R}^{\mathsf{Comp}}_{i,j}$. Each $P_k \in \mathcal{R}^{\mathsf{Comp}}_{i,j}$ does the following:
      i. Sample a seed $s_k$ and send $(\mathsf{Commit}, sid, s_k, k, \tau_{s_k})$ to $\mathcal{F}_{\mathsf{Commit}}$.
      ii. Send $(\mathsf{Open}, k, \tau_{s_k})$ to $\mathcal{F}_{\mathsf{Commit}}$ and all players obtain $s_k$ for all $k$.
      iii. Set $s = s_1 \oplus \cdots \oplus s_{n_{\mathsf{Comp}}}$, sample a uniformly random vector $\mathbf{r} = \mathcal{U}_s(p, t')$ of $t'$ elements of $\mathbb{F}_p$.
      iv. Compute the value $o^j \leftarrow \sum_{l=1}^{t'} r_l \cdot o^j_l$ and $\gamma^j_k \leftarrow \sum_{l=1}^{t'} r_l \cdot \gamma(o^j_l)_k$.
      v. Compute $\mathsf{ct}_{\gamma^j_k} = \mathsf{Encrypt}(\mathsf{pk}, \gamma^j_k; \rho^j_k)$ and $\pi^{\mathsf{Enc}}_{\gamma^j_k} \leftarrow \mathsf{Prove}((\mathsf{pk}, \mathsf{ct}_{\gamma^j_k}), (\gamma^j_k, \rho^j_k))$.
      vi. Broadcast $(o^j, \mathsf{ct}_{\gamma^j_k}, \pi^{\mathsf{Enc}}_{\gamma^j_k})$ to $\mathcal{R}^{\mathsf{Check}}_{i,1}$.
   (b) **StateReshare:** Let $\{\langle y^{i,j}_k\rangle\}_{k\in[\mathsf{int}_{i,j}]}$ be the sharings of output $\{y^{i,j}_1, \cdots, y^{i,j}_{\mathsf{int}_{i,j}}\} := \mathbf{C}_{i,j}(\mathbf{x})$ held by the parties in $\mathcal{R}^{\mathsf{Comp}}_{i,j}$ and $\mathsf{int}_{i,j}$ denotes the number of intermediate outputs of sub-circuit $\mathbf{C}_{i,j}$.
      – If $j < m$, for each $k \in [\mathsf{int}_{i,j}]$, compute resharing for $\mathcal{R}^{\mathsf{Comp}}_{i,j+1}$ as follows:
        • Each party $P_l \in \mathcal{R}^{\mathsf{Comp}}_{i,j}$ broadcasts $[y^{i,j}_k]_l + [\beta^{i,j}_k]^j_l$ and $[\gamma(y^{i,j}_k)]_l + [\eta^{i,j}_k]^j_l$ to $\mathcal{R}^{\mathsf{Comp}}_{i,j+1}$.
        • Each party $P_l \in \mathcal{R}^{\mathsf{Comp}}_{i,j+1}$ subtracts $[\beta^{i,j}_k]^{j+1}_l$ and $[\eta^{i,j}_k]^{j+1}_l$ from the values broadcast by $P_l \in \mathcal{R}^{\mathsf{Comp}}_{i,j}$, obtaining its shares of $\langle y^{i,j}_k\rangle'$.
      – If $i \neq l$ and $j = m$, execute the *State Transfer* phase in Figures 6,7 with Committees $\mathcal{R}^{\mathsf{Comp}}_{i,m}$, $\mathcal{R}^{\mathsf{Comp}}_{i+1,1}$ and $\mathcal{R}^{\mathsf{Check}}_{i,1}, \cdots, \mathcal{R}^{\mathsf{Check}}_{i,4}$ on inputs $\{\langle y^{i,m}_1\rangle, \cdots, \langle y^{i,m}_{\mathsf{int}_{i,m}}\rangle\}, \{(o^j, \mathsf{ct}_{\gamma^j_k}, \pi^{\mathsf{Enc}}_{\gamma^j_k})\}_{j\in[m]}$.
      – If $i = l$ and $j = m$, execute the *State Transfer* phase as above, but if everything verifies, skip steps for the State Update step (no era $l + 1$ exists), and execute the *Output Phase* in Figure 5 instead. If any check fails, rerun the last era with new committees.

**Output Phase:** Following successful MAC verification in the state transfer phase of $\Pi_{\mathsf{Era\text{-}MPC}}$ for final era $l$, committee $\mathcal{R}^{\mathsf{Check}}_{l,4}$ computes output share ciphertexts $\{\mathsf{ct}_{y_k}\}_{k\in[\mathsf{int}_{l,m}]}$, sends $(\mathsf{Dec}, sid, \{\mathsf{ct}_{y_k}\}_{k\in[\mathsf{int}_{l,m}]}, \mathcal{C}_{\mathsf{out}})$ to $\mathcal{F}_{\mathsf{TE}}$, and output parties receive $\{y_1, \ldots, y_{\mathsf{int}_{l,m}}\}$.

**Figure 5:** $\Pi_{\mathsf{Era\text{-}MPC}}$ — Computation Phase

The complete proof of Theorem 1 can be found in Section B.

### 3.4 Realizing $\mathcal{F}_{\mathsf{Prep}}$

Preprocessing is the one part of the construction that runs with an honest majority, and we realize it by reducing it to a single fixed arithmetic circuit $\mathbf{C}_{\mathsf{Comp}}$ evaluated through $\mathcal{F}_{\mathsf{AMPC}}$ (Figure 9), the ideal functionality for honest-majority arithmetic MPC. One evaluation yields an authenticated Beaver triple, the independent handoff masks $\beta$ and $\eta$ shared once for each of the two consecutive computation committees, and the authenticated sharing $\langle 1\rangle$ that authenticates the public term of a multiplication. Nothing is reconstructed during preprocessing: parties retain their robust output shares and forward them directly to the committees that consume them, so correlations never pass through the intervening computation committees. The two pairs delivered to checkpoint committees in encrypted form, the input masks and the era-boundary mask $R$, are not produced by a circuit at all. Each preprocessing party contributes an encrypted summand together with an argument for $\mathsf{R}_{\mathsf{Auth}}$ binding its tag to the same value, and the aggregate of the contributions that verify is authenticated by construction; this avoids homomorphically combining encrypted shares whose correctness a plaintext-knowledge argument alone would not establish.

**Input State:** At the start of this phase, each party $P_j \in \mathcal{R}^{\mathsf{Comp}}_{i,m}$ holds authenticated shares of the output wires $y_k$ of $\mathbf{C}_{i,m}$, $(y_{k,j}, \gamma(y_k)_j)$ for each $k \in [\mathsf{int}_{i,m}]$. Checkpoint committee $\mathcal{R}^{\mathsf{Check}}_{i,1}$ holds an encrypted previous state $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ and the transcript $\{o^j, (\mathsf{ct}_{\gamma^j_1}, \pi^{\mathsf{Enc}}_{\gamma^j_1}), \ldots, (\mathsf{ct}_{\gamma^j_{n_{\mathsf{Comp}}}}, \pi^{\mathsf{Enc}}_{\gamma^j_{n_{\mathsf{Comp}}}})\}_{j\in[m]}$. Set $K := \mathsf{Max}(\mathsf{int}_{i-1,m}, \mathsf{int}_{i,m})$

**Committee** $\mathcal{R}^{\mathsf{Comp}}_{i,m}$ (Commit-and-Encrypt). Each party $P_j \in \mathcal{R}^{\mathsf{Comp}}_{i,m}$ does:

1. For each output wire $k \in [\mathsf{int}_{i,m}]$, encrypt its SPDZ share: $(\mathsf{ct}_{y_{k,j}}, \mathsf{ct}_{\gamma(y_k)_j}) \leftarrow \mathsf{Encrypt}(\mathsf{pk}, (y_{k,j}, \gamma(y_k)_j))$ and compute corresponding proofs, $\pi^{\mathsf{Enc}}_{y_{k,j}}, \pi^{\mathsf{Enc}}_{\gamma(y_k)_j}$.
2. Broadcast $\{(\mathsf{ct}_{y_{k,j}}, \pi^{\mathsf{Enc}}_{y_{k,j}}, \mathsf{ct}_{\gamma(y_k)_j}, \pi^{\mathsf{Enc}}_{\gamma(y_k)_j})\}_{k\in[\mathsf{int}_{i,m}]}$ to $\mathcal{R}^{\mathsf{Comp}}_{i,m}$ and $\mathcal{R}^{\mathsf{Check}}_{i,1}$.
3. After receiving the broadcast of the ciphertexts, sample and commit to a seed $s_j$ by sending $(\mathsf{Commit}, sid, s_j, j, \tau_{s_j})$ to $\mathcal{F}_{\mathsf{Commit}}$, then open it via $(\mathsf{Open}, j, \tau_{s_j})$ to reveal it to all parties in the committee after receiving $(i, \tau_v)$ from $\mathcal{F}_{\mathsf{Commit}}$ on behalf of all parties.
4. Compute $s := \bigoplus_{j=1}^{n_{\mathsf{Comp}}} s_j$ and use it as seed to sample a uniformly random $\mathbf{r}' \leftarrow \mathcal{U}_s(p, \mathsf{int}_{i,m})$ of $\mathsf{int}_{i,m}$ elements of $\mathbb{F}_p$. Then, Broadcast $\mathbf{r}'$ to $\mathcal{R}^{\mathsf{Check}}_{i,1}$.

**Committee** $\mathcal{R}^{\mathsf{Check}}_{i,1}$ (MAC and Share Aggregation). Each party $P_j \in \mathcal{R}^{\mathsf{Check}}_{i,1}$ does:

1. Verify all received NIZK proofs corresponding to the encrypted shares and openings. If any of them fail, broadcast $(\mathsf{ct}_\alpha, \perp, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ to $\mathcal{R}^{\mathsf{Check}}_{i,2}$ and stop, else, proceed as follows.
2. Homomorphically reconstruct encrypted wire values and tags:

$$\mathsf{ct}_{y_k} := \boxplus_{l=1}^{n_{\mathsf{Comp}}} \mathsf{ct}_{y_{k,l}}, \quad \mathsf{ct}_{\gamma(y_k)} := \boxplus_{l=1}^{n_{\mathsf{Comp}}} \mathsf{ct}_{\gamma(y_k)_l}.$$

   Set the compact current state as $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}} = \{\mathsf{ct}_{y_k}, \mathsf{ct}_{\gamma(y_k)}\}_{k\in[\mathsf{int}_{i,m}]}$.
3. Obtain encrypted random SPDZ-sharing $\mathsf{ct}_R, \mathsf{ct}_{\gamma(R)}$ from $\mathcal{F}_{\mathsf{Prep}}$.
4. Compute randomized linear combinations using sampled randomness:

$$\mathsf{ct}_{y+R} := \boxplus_{k=1}^{\mathsf{int}_{i,m}} (r'_k \boxdot \mathsf{ct}_{y_k}) \boxplus \mathsf{ct}_R, \quad \mathsf{ct}_{\gamma(y+R)} := \boxplus_{k=1}^{\mathsf{int}_{i,m}} (r'_k \boxdot \mathsf{ct}_{\gamma(y_k)}) \boxplus \mathsf{ct}_{\gamma(R)}.$$

5. Compute $\mathsf{ct}_{\sigma^j} := (-o^j \boxdot \mathsf{ct}_\alpha) \boxplus \left( \boxplus_{l=1}^{n_{\mathsf{Comp}}} \mathsf{ct}_{\gamma^j_l} \right), \mathsf{ct}_\sigma := \boxplus_{j=1}^{m} \mathsf{ct}_{\sigma^j} \boxplus (-1 \boxdot \mathsf{ct}_{\gamma(y+R)})$.
6. Send $(\mathsf{Dec}, sid, \mathsf{ct}_{y+R}, \mathcal{R}^{\mathsf{Check}}_{i,2})$ to $\mathcal{F}_{\mathsf{TE}}$, broadcast $\mathsf{ct}_\alpha, \mathsf{ct}_\sigma, \mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ to $\mathcal{R}^{\mathsf{Check}}_{i,2}$.

**Figure 6:** Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$ – State Transfer Phase

**Theorem 2.** *For preprocessing (resp. computation) committees of $n_{\mathsf{Prep}}$ (resp. $n_{\mathsf{Comp}}$) parties with $t_{\mathsf{Prep}}$ (resp. $t_{\mathsf{Comp}}$) corruptions such that $n_{\mathsf{Prep}} \geq 3t_{\mathsf{Prep}} + 1$ (resp. $n_{\mathsf{Comp}} \geq t_{\mathsf{Comp}} + 1$), Protocol $\Pi_{\mathsf{Prep}}$ (Figure 19) realises $\mathcal{F}_{\mathsf{Prep}}$ (Figure 3) in the $(\mathcal{F}_{\mathsf{AMPC}})$-hybrid with guaranteed output delivery and computational security. The threshold $n_{\mathsf{Prep}} \geq 2t_{\mathsf{Prep}}+1$ suffices if $(\mathsf{Sh}, \mathsf{Rec})$ is instantiated statistically, as discussed in Remark 12.*

Protocol $\Pi_{\mathsf{Prep}}$, the circuit $\mathbf{C}_{\mathsf{Comp}}$, the protocols suitable for realizing $\mathcal{F}_{\mathsf{AMPC}}$, and the proof of Theorem 2 are given in Appendices D and E.

### 3.5 Concrete Complexity Analysis

Let $M_{i,j}$ denote the number of multiplication gates in $\mathbf{C}_{i,j}$, $\mathsf{int}_{i,j}$ the number of output wires of $\mathbf{C}_{i,j}$, and $K_i := \max(\mathsf{int}_{i-1,m}, \mathsf{int}_{i,m})$ the state-size parameter at era boundary $i$. Inside an era, the cost is that of plain SPDZ: $2M_{i,j} \cdot n_{\mathsf{Comp}}$ field-element broadcasts to evaluate $\mathbf{C}_{i,j}$, $n_{\mathsf{Comp}}$ ciphertext-proof pairs per **CheckOpen**, and $2n_{\mathsf{Comp}} \cdot \mathsf{int}_{i,j}$ field-element broadcasts per intra-era handoff, with no threshold decryption at any point. All threshold decryptions are confined to era boundaries. Crossing boundary $i$ costs 3 threshold decryptions across the four checkpoint committees (of 1, 1 and $2K_i n_{\mathsf{Comp}} + n_{\mathsf{Prep}}$ ciphertexts), $t_{\mathsf{Prep}} + 2K_i(n_{\mathsf{Comp}} - 1)$ $\Pi_{\mathsf{CoinToss}}$ calls, $2\mathsf{int}_{i,m}(n_{\mathsf{Comp}} - 1) + 2K_i(n_{\mathsf{Comp}} - 1)$ homomorphic additions, $2\mathsf{int}_{i,m}$ scalar multiplications, $n_{\mathsf{Check}}$ arguments for $\mathsf{R}_{\mathsf{Scal}}$, and $2n_{\mathsf{Comp}}\mathsf{int}_{i,m} + mn_{\mathsf{Comp}}$ NIZK verifications; the dominant term is the final decryption of $2K_i n_{\mathsf{Comp}} + n_{\mathsf{Prep}}$ ciphertexts. The public-key cost of the protocol is therefore paid once per era rather than once per committee handoff. A committee-by-committee breakdown of all four layers, including the one-time setup and input phase, is given in Appendix C.

**Committee** $\mathcal{R}_{i,2}^{\mathsf{Check}}$ (MAC Validation and Blinding). Each party $P_l \in \mathcal{R}_{i,2}^{\mathsf{Check}}$:

1. If $(\mathsf{ct}_\alpha, \bot, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ was received from $\mathcal{R}_{i,1}^{\mathsf{Check}}$, or any message required below is missing, broadcast $(\mathsf{ct}_\alpha, \bot, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ to $\mathcal{R}_{i,3}^{\mathsf{Check}}$ and stop, else, proceed as follows.
2. Receive $y' = y + R$ from $\mathcal{F}_{\mathsf{TE}}$, $\mathsf{ct}_\sigma$ from $\mathcal{R}_{i,1}^{\mathsf{Check}}$ and compute $\mathsf{ct}_c := (y' \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{ct}_\sigma$.
3. Sample $r_l \leftarrow\$ \mathbb{F}_p$, compute $\mathsf{ct}_{c,l} := (r_l \boxdot \mathsf{ct}_c) \boxplus \mathsf{Encrypt}(\mathsf{pk}, 0; s_l)$ and $\pi_l^{\mathsf{Scal}} \leftarrow \mathsf{Prove}((\mathsf{pk}, \mathsf{ct}_c, \mathsf{ct}_{c,l}), (r_l, s_l))$. Broadcast $(\mathsf{ct}_{c,l}, \pi_l^{\mathsf{Scal}})$ and $\mathsf{ct}_c$ to $\mathcal{R}_{i,3}^{\mathsf{Check}}$.
4. Broadcast $\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ to $\mathcal{R}_{i,3}^{\mathsf{Check}}$ (even if the previous steps were skipped).

**Committee** $\mathcal{R}_{i,3}^{\mathsf{Check}}$ (Aggregation). Each party in $\mathcal{R}_{i,3}^{\mathsf{Check}}$:

1. Execute $t_{\mathsf{Prep}}$ instantiations of $\Pi_{\mathsf{CoinToss}}$ such that $\mathcal{R}_{i,4}^{\mathsf{Check}}$ receives $\mathsf{ct}_{c_1}, \cdots, \mathsf{ct}_{c_t}$.
2. For each wire $k \in [K]$, execute $n_{\mathsf{Comp}} - 1$ instances of $\Pi_{\mathsf{CoinToss}}$ such that $\mathcal{R}_{i,4}^{\mathsf{Check}}$ receives masks $\{\mathsf{ct}_{r_{k,l}}, \mathsf{ct}_{\gamma(r_k)_l}\}_{l \in [n_{\mathsf{Comp}}-1]}$.
3. If $(\mathsf{ct}_\alpha, \bot, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ was received from $\mathcal{R}_{i,2}^{\mathsf{Check}}$, or any message required below is missing, broadcast $(\mathsf{ct}_\alpha, \bot, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ to $\mathcal{R}_{i,4}^{\mathsf{Check}}$ and stop, else, proceed as follows.
4. Using the base ciphertext $\mathsf{ct}_c$ received from $\mathcal{R}_{i,2}^{\mathsf{Check}}$, verify all received $\pi_l^{\mathsf{Scal}}$ and let $V$ be the set of indices whose proofs verify.
5. Compute $\mathsf{ct}_{r \cdot c} := \boxplus_{l \in V} \mathsf{ct}_{c,l}$, which encrypts $r \cdot c$ for $r := \sum_{l \in V} r_l$.
6. Send $(\mathsf{Dec}, sid, \mathsf{ct}_{r \cdot c}, \mathcal{R}_{i,4}^{\mathsf{Check}})$ to $\mathcal{F}_{\mathsf{TE}}$.
7. Broadcast $\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ to $\mathcal{R}_{i,4}^{\mathsf{Check}}$.

**Committee** $\mathcal{R}_{i,4}^{\mathsf{Check}}$ (State Update or Rollback). Each party in $\mathcal{R}_{i,4}^{\mathsf{Check}}$:

1. If $(\mathsf{ct}_\alpha, \bot, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ was received from $\mathcal{R}_{i,3}^{\mathsf{Check}}$, or if $c' = r \cdot c \neq 0$ was received from $\mathcal{F}_{\mathsf{TE}}$, set $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}} = \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ and reset the era pointer $(i, j) \leftarrow (i, 1)$; otherwise set the era pointer $(i, j) \leftarrow (i + 1, 1)$.
2. Increment the counters $\mathsf{PrepCtr} \leftarrow \mathsf{PrepCtr} + 1, \mathsf{CompCtr} \leftarrow \mathsf{CompCtr} + m, \mathsf{CheckCtr} \leftarrow \mathsf{CheckCtr} + 4$. If $(l - i) \cdot m > N_{\mathsf{Comp}} - \mathsf{CompCtr}$ or $(l - i) > N_{\mathsf{Prep}} - \mathsf{PrepCtr}$ or $(l - i) \cdot 4 > N_{\mathsf{Check}} - \mathsf{CheckCtr}$, output $\bot$ to all parties in $\mathcal{C}_{\mathsf{out}}$.
3. Select $\{\mathcal{R}_{i,j}^{\mathsf{Comp}}\}_{j \in [m]} \leftarrow \{\mathcal{C}_{\mathsf{CompCtr}}^{\mathsf{Comp}}, \ldots, \mathcal{C}_{\mathsf{CompCtr}+m-1}^{\mathsf{Comp}}\}$, $\mathcal{R}_i^{\mathsf{Prep}} \leftarrow \mathcal{C}_{\mathsf{PrepCtr}}^{\mathsf{Prep}}$, and $\{\mathcal{R}_{i,k}^{\mathsf{Check}}\}_{k \in [4]} \leftarrow \{\mathcal{C}_{\mathsf{CheckCtr}}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{CheckCtr}+1}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{CheckCtr}+2}^{\mathsf{Check}}, \mathcal{C}_{\mathsf{CheckCtr}+3}^{\mathsf{Check}}\}$ for era $i$.
4. Parse $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}} = \{\mathsf{ct}_{y_k}, \mathsf{ct}_{\gamma(y_k)}\}_{k \in [K]}$, where $K$ is the number of wires in the chosen state based on $c'$. Having received $\mathsf{ct}_\alpha, \mathsf{ct}_{c_1}, \cdots, \mathsf{ct}_{c_t}$, computes $\mathsf{ct}_{\alpha_j}$ for $\alpha_j = f(j)$ where $f(x) = \alpha + c_1 x + c_2 x^2 + \cdots + c_t x^t$.
5. For each wire $k \in [K]$, using masks $\{\mathsf{ct}_{r_{k,l}}, \mathsf{ct}_{\gamma(r_k)_l}\}_{l \in [n_{\mathsf{Comp}}-1]}$ received from $\mathcal{R}_{i,3}^{\mathsf{Check}}$, set for $l \in [n_{\mathsf{Comp}} - 1]$: $\mathsf{ct}_{[y_k]_l} := \mathsf{ct}_{r_{k,l}}$ and $\mathsf{ct}_{[\gamma(y_k)]_l} := \mathsf{ct}_{\gamma(r_k)_l}$. Compute the last-party ciphertexts homomorphically as

$$\mathsf{ct}_{[y_k]_{n_{\mathsf{Comp}}}} := \mathsf{ct}_{y_k} \boxplus \left(-1 \boxdot \boxplus_{l=1}^{n_{\mathsf{Comp}}-1} \mathsf{ct}_{r_{k,l}}\right)$$

$$\mathsf{ct}_{[\gamma(y_k)]_{n_{\mathsf{Comp}}}} := \mathsf{ct}_{\gamma(y_k)} \boxplus \left(-1 \boxdot \boxplus_{l=1}^{n_{\mathsf{Comp}}-1} \mathsf{ct}_{\gamma(r_k)_l}\right).$$

6. Send $(\mathsf{Dec}, sid, (\{(\mathsf{ct}_{[y_k]_l}, \mathsf{ct}_{[\gamma(y_k)]_l})\}_{k \in [K], l \in [n_{\mathsf{Comp}}]}, \{\mathsf{ct}_{\alpha_1}, \cdots, \mathsf{ct}_{\alpha_{n_{\mathsf{Prep}}}}\}), (\mathcal{R}_{i,1}^{\mathsf{Comp}}, \mathcal{R}_i^{\mathsf{Prep}}))$ to $\mathcal{F}_{\mathsf{TE}}$ and broadcast $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}} = \{(\mathsf{ct}_{[y_k]_l}, \mathsf{ct}_{[\gamma(y_k)]_l})\}_{k \in [K], l \in [n_{\mathsf{Comp}}]}$ and $\mathsf{ct}_\alpha$ to $\mathcal{R}_{i,1}^{\mathsf{Check}}$.

**Figure 7:** Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$ – State Transfer Phase Continued

# 4 Optimizations and Variants

## 4.1 Transferring Correlated Randomness via a Bulletin Board

The first variant we explore is using a public bulletin board to maintain encrypted correlated randomness for future eras. While requiring a public bulletin board is a stronger setup assumption, the advantage of this approach is twofold: 1. more flexible preprocessing with lower amortized cost; 2. public verifiability, which we will introduce in the following section. In this variant, a single preprocessing committee can generate correlated randomness for multiple future eras whose computation committees are not yet online without resorting to expensive resharing to transfer this randomness across committees.

The main challenge to be solved in this setting is how to efficiently transfer correlated randomness generated by a given preprocessing committee to a computation committee that only becomes online at some arbitrary point in the future. In order to do so, we adapt an idea from the literature on Encryption to the Future [18] to obtain a protocol where only a secret key must be reshared across checkpoint committees. The correlated randomness to be transferred only needs to be published by the preprocessing committee once and retrieved by the receiving computation committee when needed, regardless of how far in the future it will be used.

This approach is based on an Identity Based Encryption (IBE) scheme where the master secret and public key pair is generated via a Distributed Key Generation protocol executed by the initial checkpoint committees. Each checkpoint committee then reshares the master secret key to subsequent committees in order to keep it available. The current checkpoint committee can execute a protocol to generate a key for a publicly known ID from the master secret key shares. We capture such a scheme in Functionality $\mathcal{F}_{\mathsf{IBE}}$ (Figure 12) and discuss how to realize it in Appendix A.7.

Each party $P_i$ in a computation committee $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ is deterministically assigned an ID $\mathsf{id}_{i,\mathsf{ctr}}$. After generating correlated randomness for $P_i$, each party $P_j$ in the preprocessing committee encrypts its shares with a symmetric encryption scheme under a fresh secret key $\mathsf{sk}_{i,j,\mathsf{ctr}}$ obtaining a ciphertext $\mathsf{ct}_{i,j,\mathsf{ctr}}$. Moreover, $P_j$ encrypts $\mathsf{sk}_{i,\mathsf{ctr}}$ with the IBE scheme under $\mathsf{id}_{i,\mathsf{ctr}}$, obtaining a ciphertext $\mathsf{ct}'_{i,j,\mathsf{ctr}}$. Both $\mathsf{ct}_{i,j,\mathsf{ctr}}, \mathsf{ct}'_{i,j,\mathsf{ctr}}$ are published in the public bulletin board. When $P_i$ becomes online, the current checkpoint committee executes the ID key generation protocol so that $P_i$ obtains the decryption key $\mathsf{sk}_{i,\mathsf{ctr}}$ corresponding to $\mathsf{id}_{i,\mathsf{ctr}}$. Now, $P_i$ can retrieve every pair $\mathsf{ct}_{i,j,\mathsf{ctr}}, \mathsf{ct}'_{i,j,\mathsf{ctr}}$ from the bulletin board, decrypt $\mathsf{ct}'_{i,j,\mathsf{ctr}}$ to obtain $\mathsf{sk}_{i,j,\mathsf{ctr}}$ and then use it to decrypt $\mathsf{ct}_{i,j,\mathsf{ctr}}$ to obtain $P_j$'s share of the correlated randomness.

### 4.2 Public Verifiability via a Bulletin Board

A further optimization that can be directly obtained when using a public bulletin board is achieving public verifiability by publishing the latest encrypted computation state verified by the checkpoint committees, as well as the result of this verification. In this variant, all broadcast communication is done via the public bulletin board, meaning that the last computation committee of an era publishes encryptions of their output shares and corresponding MACs. After verifying the validity of these shares, all parties in the last checkpoint committee in the era additionally publish the result of the verification. A third party can check that a given state was valid by checking that a majority of the checkpoint committee published a message confirming that the verification succeeded. Alternatively, if we treat the threshold decryption scheme in a non-black box manner, the decryption protocol revealing the result of the MAC verification can be executed by the checkpoint committees by broadcasting messages on the public bulletin board. For many natural threshold decryption protocols, this allows parties to verify the validity of the partial decryptions and recover the decrypted result of the MAC verification.

We formalize a suitable Eras MPC functionality $\mathcal{F}_{\mathsf{Era-PV}}$ with public verifiability in Definition 3, describe the modified protocol $\Pi_{\mathsf{Era-PV}}$ in Definition 4 and formalize its security in Theorem 5.

**Definition 3** ($\mathcal{F}_{\mathsf{Era-PV}}$)**.** *Functionality $\mathcal{F}_{\mathsf{Era-PV}}$ works as $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ and adds the following interface:*

- ***Public Verifiability:*** *Upon receiving* $(\mathsf{Verify}, \mathsf{varid})$ *from any party $P_i$, check that identifier* varid *exists and is associated to a value stored in Era $i'$ such that $i' < i$. If the check passes, set $v = 1$, else, set $v = 0$. Output* $(\mathsf{Verify}, \mathsf{varid}, v)$ *to $P_i$.*

**Definition 4 (Protocol $\Pi_{\mathsf{Era-PV}}$).** *Protocol $\Pi_{\mathsf{Era-PV}}$ is obtained by instantiating Protocol $\Pi_{\mathsf{Era}}$-MPC in the the $(\mathcal{F}_{\mathsf{Prep}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{TE}}, \mathcal{F}_{\mathsf{BB}})$-hybrid model with the following modifications:*

- *For every message $m$ that is broadcast by a party $P_i$ in $\Pi_{\mathsf{Era}}$-MPC, party $P_i$ sends* $(\mathsf{Write}, sid, m)$ *to $\mathcal{F}_{\mathsf{BB}}$.*
- *For every message $m$ that a party $P_i$ expects to receive from party $P_j$ in $\Pi_{\mathsf{Era}}$-MPC, party $P_i$ sends* $(\mathsf{Read}, sid)$ *to $\mathcal{F}_{\mathsf{BB}}$ and retrieves $m$ from the list $(\cdot, \cdot, m)$ returned by $\mathcal{F}_{\mathsf{BB}}$.*
- *In Step 1 of the State Update or Rollback phase for $\mathcal{C}_{i,4}^{\mathsf{Check}}$ in $\Pi_{\mathsf{StateTransfer}}$, if $(\mathsf{ct}_\alpha, \perp, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ is received from $\mathcal{C}_{i,4}^{\mathsf{Check}}$ or $c \neq 0$ is received from $\mathcal{F}_{\mathsf{TE}}$, all parties in $\mathcal{C}_{i,4}^{\mathsf{Check}}$ sets $v = 0$. Otherwise, all parties in $\mathcal{C}_{i,4}^{\mathsf{Check}}$ set $v = 1$. All parties in $\mathcal{C}_{i,4}^{\mathsf{Check}}$ send* $(\mathsf{Write}, sid, (\mathsf{Verification}, \mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}, v))$ *to $\mathcal{F}_{\mathsf{BB}}$.*
- *On input* $(\mathsf{Verify}, \mathsf{varid})$ *where* varid *corresponds to a computation state $\mathsf{ct}_{\mathsf{st}}$, any party $P_i$ sends* $(\mathsf{Read}, sid)$ *to $\mathcal{F}_{\mathsf{BB}}$ and checks that $\mathsf{ct}_{\mathsf{st}}$ corresponds to one the states published by one of the computation committees $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ of a previous era in the list $(\cdot, \cdot, m)$ returned by $\mathcal{F}_{\mathsf{BB}}$. If yes, $P_i$ checks that at least $t_{\mathsf{Check}} + 1$ parties of the corresponding committee $\mathcal{C}_{j,4}^{\mathsf{Check}}$ published a message* $(\mathsf{Verification}, \mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}, 1)$ *in $\mathcal{F}_{\mathsf{BB}}$. If both checks succeed, set $v = 1$, else, set $v = 0$. Output* $(\mathsf{Verify}, \mathsf{varid}, v)$.

**Theorem 5.** *For computation committees of $n_{\mathsf{Comp}}$ parties with $t_{\mathsf{Comp}}$ corruptions, preprocessing committees of $n_{\mathsf{Prep}}$ parties with $t_{\mathsf{Prep}}$ corruptions and checkpoint committees of $n_{\mathsf{Check}}$ parties with $t_{\mathsf{Check}}$ corruptions such that $n_{\mathsf{Comp}} \geq t_{\mathsf{Comp}} + 1$, $n_{\mathsf{Prep}} \geq 2t_{\mathsf{Prep}} + 1$ and $n_{\mathsf{Check}} \geq 2t_{\mathsf{Check}} + 1$, Protocol $\Pi_{\mathsf{Era-PV}}$ (Definition 4) UC-realizes $\mathcal{F}_{\mathsf{Era-PV}}$ (Definition 3) in the $(\mathcal{F}_{\mathsf{Prep}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{TE}}, \mathcal{F}_{\mathsf{BB}})$-hybrid model with security against a static adversary.*

*Proof.* (Sketch) The proof follows mostly from the proof of Theorem 1, as replacing the broadcasts with communication over the public bulletin board functionality $\mathcal{F}_{BB}$ preserves the original properties of Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$, since the adversary learns no extra information. The main difference in Protocol $\Pi_{\mathsf{Era\text{-}MPC}}$ is in the public verifiability steps, the crucially rely on the fact that $n_{\mathsf{Check}} \geq 2t_{\mathsf{Check}} + 1$ and that $\mathcal{F}_{BB}$ authenticates the messages published by each party. Notice that the adversary cannot publish messages $(\mathsf{Verification}, \mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}, 1)$ from more than $t_{\mathsf{Check}}$ different parties any checkpoint committee confirming the successful verification of an invalid state $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}$. Hence, the simulation becomes trivial, as the simulator can simply follow the steps of the simulator of $\Pi_{\mathsf{Era\text{-}MPC}}$ for most of the protocol (although publishing broadcasted messages on $\mathcal{F}_{BB}$), execute the public verification steps exactly as an honest party and simulate $\mathcal{F}_{BB}$ towards the adversary exactly as it is described.

### 4.3 Realizing Reactive Functionalities

While we describe a protocol for securely evaluating a fixed circuit with a fixed input phase, we note that it is straightforward to extend it to the case of reactive computation. The boundary between eras can be used to not only verify the current state of the computation but to provide outputs and receive new inputs. Instead of immediately proceeding to the next era when the last checkpoint committee obtains the result of the intermediate computation state verification, this committee can decrypt specific wire values and reveal them to arbitrary sets of parties. New inputs can be provided by executing the input phase after the outputs are revealed and before the next era starts. It is also possible to adjust the sub-circuits to be evaluated in the next era according to the outputs revealed after the previous era by postponing the preprocessing for the next era until after new inputs and new circuits are provided.

### 4.4 Post-Compromise Integrity without Erasures

Although we formally state our results against a static adversary, one may consider the case where a preprocessing committee or computation committee becomes corrupted over the assumed corruption threshold, exposing the MAC key $\alpha$. If such corruption happens while the committee is active, it is impossible to preserve neither the integrity nor the privacy of the computation. However, if the corruption happens after the committee is inactive, it is possible to preserve at least integrity without resorting to secure erasures.

To ensure that a compromise of the global MAC key $\alpha$ used in era $i$ does not compromise the integrity of future eras, the protocol performs a key rotation to a fresh $\alpha'$ during the state transfer. This is achieved by staggering the knowledge of the key and the state-mask across a sequence of committees:

1. **Key Sampling:** A checkpoint committee samples a fresh key $\alpha'$ via $\Pi_{\mathsf{CoinToss}}$, resulting in $\mathsf{ct}_{\alpha'}$. It then uses $\Pi_{\mathsf{CoinToss}}$ to create shares of $\alpha'$ that are opened towards a future preprocessing committee $\mathcal{C}^{\mathsf{Prep}}_{i'}$.
2. **Preprocessing:** Similar to the **PrepInit** phase, $\mathcal{C}^{\mathsf{Prep}}_{i'}$ computes $\mathsf{ct}_r$ and $\mathsf{ct}_{\alpha' \cdot r}$ for a random $r$ chosen inside the functionality and provides these to a subsequent checkpoint committee. This checkpoint committee uses the aggregate state $\mathsf{ct}_y$ and the mask $\mathsf{ct}_r$ to reveal $y + r$ to a future checkpoint committee.
3. **State Handoff:** Using $\mathsf{ct}_{\alpha'}, \mathsf{ct}_{\alpha' \cdot r}$, and the plaintext $m = y + r$, committee $\mathcal{C}^{\mathsf{Check}}_{i,4}$ homomorphically computes the new tag:

$$\mathsf{ct}_{\gamma'(y)} := (m \boxdot \mathsf{ct}_{\alpha'}) \boxplus (-1 \boxdot \mathsf{ct}_{\alpha' \cdot r})$$

The resulting pair $(\mathsf{ct}_y, \mathsf{ct}_{\gamma'(y)})$ forms the new era state, which is then reshared to $\mathcal{C}^{\mathsf{Comp}}_{i+1,1}$ using the state transfer procedure used to reshare $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}$.

This approach ensures post-compromise integrity because the adversary, even if in possession of the old $\alpha$, cannot forge tags for the new era authenticated under the secret $\alpha'$. Furthermore, since the rotation utilizes a masked value $y + r$ and the knowledge of $r$ and $\alpha'$ is distributed across disjoint committees with honest majorities, the plaintext state $y$ is never exposed.

### 4.5 Guaranteed Output Delivery through Fallback.

While our optimistic execution protocol achieves security with abort, the protocol can be extended to provide Guaranteed Output Delivery (GOD) if the adversary causes persistent disruptions. If the computation fails to progress after a threshold of aborted eras, the checkpoint committee initiates a fallback mechanism. The committee uses the $\mathcal{F}_{\mathsf{TE}}$ functionality to decrypt the aggregate state $\mathsf{ct}_{y_k}$ directly into secret shares towards parties that execute an MPC protocol with GOD. Notice that this can be a standard MPC protocol or any dynamic MPC protocol that provides GOD. This allows the system to bypass the disruption and ensure the remaining computation is successfully executed, leveraging the checkpoint committee as a bridge between the optimistic and robust execution modes.

**Definition 6 ($\mathcal{F}_{\mathsf{Era-GOD}}$).** *Functionality $\mathcal{F}_{\mathsf{Era-GOD}}$ works as $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ except for the following differences:*

- ***Parties:*** *The functionality also interacts with a pool of honest majority computation committees $\mathcal{P}_{\mathsf{Comp-GOD}}$, who carry out the GOD computation.*
- ***Era Abort:*** *Upon receiving $(sid, \mathsf{EraAbort}, i)$ from the adversary $\mathcal{S}$, set $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and $(i, j) \leftarrow (i, 1)$, and if $(l - i) \cdot m > N_{\mathsf{Comp}} - \mathsf{ctr}$, ignore further $(sid, \mathsf{EraAbort}, i)$ messages and send $(sid, \mathsf{OptimisticFail})$ to $\mathcal{C}_1^{\mathsf{Comp-GOD}}$ (roles assigned to honest majority committees in $\mathcal{P}_{\mathsf{Comp-GOD}}$).*
- ***GOD-Evaluate:*** *Upon receiving $(sid, \mathsf{EvaluateGOD})$ from a majority of the parties in $\mathcal{C}_1^{\mathsf{Comp-GOD}}$, evaluate the remaining circuits $\{\mathbf{C}_{i,j}\}_{i\in[k,l],j\in[m]}$ and send $(sid, \mathsf{OutputSuccess})$ to $\mathcal{C}_{\mathsf{out}}$.*

**Definition 7 (Protocol $\Pi_{\mathsf{Era-GOD}}$).** *Protocol $\Pi_{\mathsf{Era-GOD}}$ works exactly as $\Pi_{\mathsf{Era}}$-MPC except for the following differences:*

- ***Setup:*** *Besides the setup for $\Pi_{\mathsf{Era}}$-MPC, $\Pi_{\mathsf{Era-GOD}}$ also uses an MPC functionality with GOD $\mathcal{F}_{\mathsf{AMPC}}$, which is used as a fallback after the optimistic approach of $\Pi_{\mathsf{Era}}$-MPC fails.*
- ***Step 2 of Committee $\mathcal{C}_{i,4}^{\mathsf{Check}}$ (State Update or Rollback) in the State Transfer Phase:*** *If $(l - i) \cdot m > N_{\mathsf{Comp}} - \mathsf{CompCtr}$ or $(l - i) > N_{\mathsf{Prep}} - \mathsf{PrepCtr}$ or $(l - i) \cdot 4 > N_{\mathsf{Check}} - \mathsf{CheckCtr}$, send to committee $\mathcal{C}_1^{\mathsf{Comp-GOD}}$ fresh $t$-out-of-$n$ robust shares of the previous valid state in $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ using the same procedure as in the resharing of $\mathsf{ct}_\alpha$ to $\mathcal{C}_{i+1}^{\mathsf{Prep}}$.*
- ***GOD evaluation:*** *$\mathcal{C}_1^{\mathsf{Comp-GOD}}$ (or a sequence of committees $\mathcal{C}_1^{\mathsf{Comp-GOD}}, \ldots, \mathcal{C}_n^{\mathsf{Comp-GOD}}$ in case of a dynamic committee protocol) uses $\mathcal{F}_{\mathsf{AMPC}}$ to evaluate the following circuits: 1. a circuit reconstructing the values of the wires from the shares received from $\mathcal{C}_{i,4}^{\mathsf{Check}}$; 2. the remaining circuits $\{\mathbf{C}_{i,j}\}_{i\in[k,l],j\in[m]}$; 3. evaluate a circuit that outputs verifiable shares of the output of $\mathbf{C}_{l,m}$. The last GOD computation committee $\mathcal{C}_n^{\mathsf{Comp-GOD}}$ sends their shares of the output to $\mathcal{C}_{\mathsf{out}}$, who reconstructs the result.*

**Theorem 8.** *For computation committees of $n_{\mathsf{Comp}}$ parties with $t_{\mathsf{Comp}}$ corruptions, preprocessing committees of $n_{\mathsf{Prep}}$ parties with $t_{\mathsf{Prep}}$ corruptions, checkpoint committees of $n_{\mathsf{Check}}$ parties with $t_{\mathsf{Check}}$ and GOD computation committees of $n_{\mathsf{CompGOD}}$ parties with $t_{\mathsf{CompGOD}}$ corruptions such that $n_{\mathsf{Comp}} \geq t_{\mathsf{Comp}} + 1$, $n_{\mathsf{Prep}} \geq 2t_{\mathsf{Prep}} + 1$, $n_{\mathsf{Check}} \geq 2t_{\mathsf{Check}} + 1$ and $n_{\mathsf{CompGOD}} \geq 2t_{\mathsf{CompGOD}}$, Protocol $\Pi_{\mathsf{Era-GOD}}$ (Definition 4) UC-realizes $\mathcal{F}_{\mathsf{AMPC}}$ (Definition 9) in the $(\mathcal{F}_{\mathsf{Prep}}, \mathcal{F}_{\mathsf{Commit}}, \mathcal{F}_{\mathsf{TE}}, \mathcal{F}_{\mathsf{BB}}, \mathcal{F}_{\mathsf{AMPC}})$-hybrid model with security static adversaries.*

*Proof.* The proof follows trivially from the security of $\Pi_{\mathsf{Era\text{-}MPC}}$ use of $\mathcal{F}_{\mathsf{AMPC}}$ to evaluate the remaining circuits after the original $\Pi_{\mathsf{Era\text{-}MPC}}$ fails.

## 5 Communication Lower Bounds

We establish communication lower bounds for the handover phase between different committees. These bounds show that two subsequent committees of sizes $n$ and $n'$ must communicate $O(n \cdot n' \|w|)$

per wire where $|w|$ is the size of the representation of a wire value. In current protocols, this is reflected in the use of canonical resharing protocols that match these bounds. This fundamental limit motivates the use of protocols where each committee executes more than one round, amortizing this handover cost over multiple rounds of computation. The bounds are formalized and proven in the theorems below.

**Theorem 9.** *Let $\Pi$ be an MPC protocol realizing $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ with perfect security and guaranteed output delivery where a committee $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ of $n = t+1$ parties $P_i$, for $i = 1, \ldots, n$ hold shares $s_i$ that reconstruct the value of a wire, such that a committee $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ of $n' = t' + 1$ parties $P_j$ for $j = 1, \ldots, n'$ such that $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}} \cap \mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}} = \emptyset$ hold shares $s_j$ that reconstruct to the same wire value, where $t, t'$ parties are passively corrupted. $\Pi$ has communication complexity $O(n \cdot n' \cdot |w|)$, where $|w|$ is the size of the wire value representation.*

*Proof (of Theorem 9).* Let committee $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ of $n$ parties $P_i$, for $i = 1, \ldots, n$ have $n+1 = t$ corrupted parties $C_1, \ldots, C_t$ holding shares $s_{c,1}, \ldots, s_{c,t}$ when the protocol starts. Let committee $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ of $n'$ parties $P_j$, for $j = 1, \ldots, n'$ have $n'+1 = t'$ corrupted parties $C'_1, \ldots, C'_t$ holding shares $s_{c',1}, \ldots, s_{c',t'}$ after the protocol execution. Notice that parties in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ that only receive messages from $t$ parties from $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ cannot reconstruct the value of the wire with probability 1 while realizing $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$. This is because if shares from only $t$ parties from $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ completely determined the wire value, the adversary would learn this value, contradicting the fact that $\Pi$ realizes $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$.

The adversary corrupting $t$ parties of $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ may send corrupted messages to $n'$ parties of $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$, out of which it corrupts $t'$ parties. However, the adversary cannot learn the value of the wire from the $t'$ internal states of corrupted parties of $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ (including the messages they receive from $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$) although it already knows the internal states of the $t$ parties corrupted in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$. If the adversary corrupting $t$ parties of $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ and $t'$ parties of $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ could learn the value of the wire from the messages exchanged between the parties in both committees, the protocol would not realize the prescribed functionality.

Every honest party in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ must exchange a message of the same length of the secret with the every honest party in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$. Otherwise, the adversary corrupting $t$ parties of $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ may learn the remaining information to recover the value of the wire by corrupting $t'$ parties in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$. Notice that if not all $n$ parties in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ exchange messages with all $n'$ parties in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$, the adversary may reconstruct the wire value by learning the messages received by the corrupted parties who received a message from parties in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$.

**Theorem 10.** *Let $\Pi$ be an MPC protocol realizing $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ with perfect security and guaranteed output delivery where a committee $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ of $n = 2t + 1$ parties $P_i$, for $i = 1, \ldots, n$ hold shares $s_i$ that reconstruct the value $w$ of a wire, such that a committee $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ of $n' = 2t' + 1$ parties $P_j$ for $j = 1, \ldots, n'$ such that $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}} \cap \mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}} = \emptyset$ hold shares $s_j$ that reconstruct to the same wire value, where $t, t'$ parties are actively corrupted. $\Pi$ has communication complexity $O(n \cdot n' \cdot |w|)$, where $|w|$ is the size of the wire value representation.*

*Proof (of Theorem 10).* Let committee $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ of $n$ parties $P_i$, for $i = 1, \ldots, n$ have $n+1 = t$ corrupted parties $C_1, \ldots, C_t$ holding shares $s_{c,1}, \ldots, s_{c,t}$. Let committee $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ of $n'$ parties $P_j$, for $j = 1, \ldots, n'$ have $n'+1 = t'$ corrupted parties $C'_1, \ldots, C'_t$ holding shares $s_{c',1}, \ldots, s_{c',t'}$ after the protocol execution.

Assume that the adversary does not send messages from $C_1, \ldots, C_t$ to the parties in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$, but receives the messages sent by honest parties in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ to $C'_1, \ldots, C'_t$. Because the honest parties in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ must be able to reconstruct the wire value even if the adversary does not cooperate, the $t + 1$ messages they receive from the honest parties in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ must completely determine the secret. However, the $t$ messages that $C'_1, \ldots, C'_t$ receive must not reveal any information about the wire value, even when combined with the $s_{c,1}, \ldots, s_{c,t}$ held by $C_1, \ldots, C_t$.

It is clear that all honest parties in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ must send messages to all parties in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$. Less than $t+1$ shares held by $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ do not contain any information about the wire value. Otherwise, the adversary would learn something about it. Hence, only if all $t + 1$ honest parties in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ send messages to $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$ can the receiving parties learn enough information to reconstruct the wire. If honest parties in $\mathcal{C}_{\mathsf{ctr}}^{\mathsf{Comp}}$ only sent messages to less than $n'$ parties in $\mathcal{C}_{\mathsf{ctr+1}}^{\mathsf{Comp}}$, there is a non-zero probability than less than $t'+1$ honest parties would receive messages that determine the secret. Hence, in that case, there is a non-zero probability that the adversary can abort the next step of the protocol or the wire value

reconstruction by aborting parties $C'_1, \ldots, C'_t$, resulting in less than $t'+1$ messages being sent by $\mathcal{C}^{\mathsf{Comp}}_{\mathsf{ctr}+1}$.

It must be possible for $t'+1$ parties in $\mathcal{C}^{\mathsf{Comp}}_{\mathsf{ctr}+1}$ to reconstruct the wire value $w$ from the messages they receive from $t+1$ parties in $\mathcal{C}^{\mathsf{Comp}}_{\mathsf{ctr}}$. Since $t$ sending parties in $\mathcal{C}^{\mathsf{Comp}}_{\mathsf{ctr}}$ and $t'$ receiving parties in $\mathcal{C}^{\mathsf{Comp}}_{\mathsf{ctr}+1}$ are corrupt, the only message $m$ the adversary does not know, is the one sent from the single honest sending party in $\mathcal{C}^{\mathsf{Comp}}_{\mathsf{ctr}}$ to the single honest receiving party in $\mathcal{C}^{\mathsf{Comp}}_{\mathsf{ctr}+1}$. The uncertainty on $w$ is maximal given the adversary's view, but drops to 0 if we add $m$, so it follows from the chain rule for entropy that $m$ must be at least as large as $w$. This holds for any pair of honest players.

## References


1. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: SCALES - MPC with small clients and larger ephemeral servers. In: Kiltz, E., Vaikuntanathan, V. (eds.) TCC 2022, Part II. LNCS, vol. 13748, pp. 502–531. Springer, Cham (Nov 2022)
2. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: Malicious security for SCALES - outsourced computation with ephemeral servers. In: Reyzin, L., Stebila, D. (eds.) CRYPTO 2024, Part IX. LNCS, vol. 14928, pp. 3–38. Springer, Cham (Aug 2024)
3. Acharya, A., Hazay, C., Kolesnikov, V., Prabhakaran, M.: Towards building efficient SCALES protocols. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part V. LNCS, vol. 16249, pp. 35–67. Springer, Singapore (Dec 2025)
4. Acharya, A., Meyer, P., Ravi, D., Satish, R.: Sublinear-communication layered MPC from HSS. Cryptology ePrint Archive, Paper 2026/1445 (2026), `https://eprint.iacr.org/2026/1445`
5. Badertscher, C., Gazi, P., Kiayias, A., Russell, A., Zikas, V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) ACM CCS 2018. pp. 913–930. ACM Press (Oct 2018)
6. Badertscher, C., Maurer, U., Tschudi, D., Zikas, V.: Bitcoin as a transaction ledger: A composable treatment. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 324–356. Springer, Cham (Aug 2017)
7. Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: Secure MPC: Laziness leads to GOD. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 120–150. Springer, Cham (Dec 2020)
8. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988)
9. Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., Reyzin, L.: Can a public blockchain keep a secret? In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part I. LNCS, vol. 12550, pp. 260–290. Springer, Cham (Nov 2020)
10. Benhamouda, F., Halevi, S., Krawczyk, H., Miao, A., Rabin, T.: Threshold cryptography as a service (in the multiserver and YOSO models). In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 323–336. ACM Press (Nov 2022)
11. Bhadauria, R., yu Chiang, J.H., Ravi, D., Sternad, J., Yakoubov, S.: Minimizing the use of the honest majority in YOSO MPC with guaranteed output delivery. Cryptology ePrint Archive, Report 2024/2059 (2024), `https://eprint.iacr.org/2024/2059`
12. Bienstock, A., Escudero, D., Polychroniadou, A.: On linear communication complexity for (maximally) fluid MPC. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part I. LNCS, vol. 14081, pp. 263–294. Springer, Cham (Aug 2023)
13. Bienstock, A., Escudero, D., Polychroniadou, A.: Perfectly secure fluid MPC with abort and linear communication complexity. CiC 1(4), 28 (2024)
14. Blum, E., Leung, D., Loss, J., Katz, J., Rabin, T.: Analyzing the real-world security of the algorand blockchain. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023. pp. 830–844. ACM Press (Nov 2023)
15. Boneh, D., Franklin, M.K.: Identity-based encryption from the Weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Berlin, Heidelberg (Aug 2001)
16. Braun, L., Castagnos, G., Damgård, I., Laguillaumie, F., Melissaris, K., Orlandi, C., Tucker, I.: An improved threshold homomorphic cryptosystem based on class groups. In: Galdi, C., Phan, D.H. (eds.) SCN 24, Part II. LNCS, vol. 14974, pp. 24–46. Springer, Cham (Sep 2024)
17. Braun, L., Damgård, I., Orlandi, C.: Secure multiparty computation from threshold encryption based on class groups. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part I. LNCS, vol. 14081, pp. 613–645. Springer, Cham (Aug 2023)
18. Campanelli, M., David, B., Khoshakhlagh, H., Konring, A., Nielsen, J.B.: Encryption to the future - A paradigm for sending secret messages to future (anonymous) committees. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part III. LNCS, vol. 13793, pp. 151–180. Springer, Cham (Dec 2022)

19. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
20. Canetti, R., Kolby, S., Ravi, D., Soria-Vazquez, E., Yakoubov, S.: Taming adaptivity in YOSO protocols: The modular way. In: Rothblum, G.N., Wee, H. (eds.) TCC 2023, Part II. LNCS, vol. 14370, pp. 33–62. Springer, Cham (Nov / Dec 2023)
21. Cascudo, I., David, B.: Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part V. LNCS, vol. 14655, pp. 216–248. Springer, Cham (May 2024)
22. Cascudo, I., David, B., Garms, L., Konring, A.: YOLO YOSO: Fast and simple encryption and secret sharing in the YOSO model. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part I. LNCS, vol. 13791, pp. 651–680. Springer, Cham (Dec 2022)
23. Castagnos, G., Laguillaumie, F., Tucker, I.: Threshold linearly homomorphic encryption on $\mathbf{Z}/2^k\mathbf{Z}$. In: Agrawal, S., Lin, D. (eds.) ASIACRYPT 2022, Part II. LNCS, vol. 13792, pp. 99–129. Springer, Cham (Dec 2022)
24. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC. pp. 11–19. ACM Press (May 1988)
25. Choudhuri, A.R., Goel, A., Green, M., Jain, A., Kaptchuk, G.: Fluid MPC: Secure multiparty computation with dynamic participants. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 94–123. Springer, Cham, Virtual Event (Aug 2021)
26. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–299. Springer, Berlin, Heidelberg (May 2001)
27. Damgård, I., Escudero, D., Polychroniadou, A.: Phoenix: Secure computation in an unstable network with dropouts and comebacks. In: Chung, K.M. (ed.) ITC 2023. LIPIcs, vol. 267, pp. 7:1–7:21. Schloss Dagstuhl (Jun 2023)
28. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Berlin, Heidelberg (Sep 2013)
29. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Berlin, Heidelberg (Aug 2012)
30. David, B., Deligios, G., Goel, A., Ishai, Y., Konring, A., Kushilevitz, E., Liu-Zhang, C.D., Narayanan, V.: Perfect MPC over layered graphs. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Part I. LNCS, vol. 14081, pp. 360–392. Springer, Cham (Aug 2023)
31. David, B., Lavagnino, L., Pagnin, E., Wagner, P.S.: Tail-hammer: Optimized statistics for anonymous committees and applications. Cryptology ePrint Archive, Paper 2026/1101 (2026), `https://eprint.iacr.org/2026/1101`, to appear at SCN 20206
32. David, B., Magri, B., Matt, C., Nielsen, J.B., Tschudi, D.: GearBox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 683–696. ACM Press (Nov 2022)
33. David, B., Mondal, A., Satish, R.: Rumors MPC: GOD for dynamic committees, low communication via constant-round chat. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part V. LNCS, vol. 16249, pp. 102–132. Springer, Singapore (Dec 2025)
34. Deligios, G., Konring, A., Liu-Zhang, C.D., Narayanan, V.: Statistical layered MPC. In: Boyle, E., Mahmoody, M. (eds.) TCC 2024, Part IV. LNCS, vol. 15367, pp. 362–394. Springer, Cham (Dec 2024)
35. Eldefrawy, K., Hwang, S., Ostrovsky, R., Yung, M.: Communication-efficient (proactive) secure computation for dynamic general adversary structures and dynamic groups. In: Galdi, C., Kolesnikov, V. (eds.) SCN 20. LNCS, vol. 12238, pp. 108–129. Springer, Cham (Sep 2020)
36. Eldefrawy, K., Lepoint, T., Leroux, A.: Communication-efficient proactive MPC for dynamic groups with dishonest majorities. In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, vol. 13269, pp. 565–584. Springer, Cham (Jun 2022)
37. Erwig, A., Faust, S., Riahi, S.: Large-scale non-interactive threshold cryptosystems through anonymity. Cryptology ePrint Archive, Report 2021/1290 (2021), `https://eprint.iacr.org/2021/1290`
38. Escudero, D., Goyal, V., Polychroniadou, A., Song, Y.: TurboPack: Honest majority MPC with constant online communication. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 951–964. ACM Press (Nov 2022)
39. Escudero, D., Goyal, V., Polychroniadou, A., Song, Y., Weng, C.: SuperPack: Dishonest majority MPC with constant online communication. In: Hazay, C., Stam, M. (eds.) EUROCRYPT 2023, Part II. LNCS, vol. 14005, pp. 220–250. Springer, Cham (Apr 2023)
40. Escudero, D., Masserova, E., Polychroniadou, A.: Towards scalable YOSO MPC via packed secret-sharing. In: Hanaoka, G., Yang, B.Y. (eds.) ASIACRYPT 2025, Part V. LNCS, vol. 16249, pp. 68–101. Springer, Singapore (Dec 2025)

41. Fischlin, M.: Communication-efficient non-interactive proofs of knowledge with online extractors. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 152–168. Springer, Berlin, Heidelberg (Aug 2005)
42. Fitzi, M., Hirt, M., Maurer, U.M.: Trading correctness for privacy in unconditional multi-party computation (extended abstract). In: Krawczyk, H. (ed.) CRYPTO'98. LNCS, vol. 1462, pp. 121–136. Springer, Berlin, Heidelberg (Aug 1998)
43. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: You only speak once - secure MPC with stateless ephemeral roles. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 64–93. Springer, Cham, Virtual Event (Aug 2021)
44. Gentry, C., Halevi, S., Magri, B., Nielsen, J.B., Yakoubov, S.: Random-index PIR and applications. In: Nissim, K., Waters, B. (eds.) TCC 2021, Part III. LNCS, vol. 13044, pp. 32–61. Springer, Cham (Nov 2021)
45. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987)
46. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ATLAS: Efficient and scalable MPC in the honest majority setting. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 244–274. Springer, Cham, Virtual Event (Aug 2021)
47. Guo, Y., Pass, R., Shi, E.: Synchronous, with a chance of partition tolerance. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 499–529. Springer, Cham (Aug 2019)
48. Hirt, M., Maurer, U.M., Przydatek, B.: Efficient secure multi-party computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Berlin, Heidelberg (Dec 2000)
49. Kolby, S., Ravi, D., Yakoubov, S.: Constant-round YOSO MPC without setup. CiC 1(3), 30 (2024)
50. Kolby, S., Roy, L., Sternad, J., Yakoubov, S.: Information-theoretic random-index PIR. In: Gilboa, N. (ed.) ITC 2025. LIPIcs, vol. 343, pp. 5:1–5:15. Schloss Dagstuhl (Aug 2025)
51. Liu-Zhang, C.D., Masserova, E., Ribeiro, J., Soni, P., Thyagarajan, S.A.K.: Efficient distributed randomness generation from minimal assumptions where PArties speak sequentially once. In: Fehr, S., Fouque, P.A. (eds.) EUROCRYPT 2025, Part V. LNCS, vol. 15605, pp. 176–206. Springer, Cham (May 2025)
52. Nielsen, J.B., Ribeiro, J.L., Obremski, M.: Public randomness extraction with ephemeral roles and worst-case corruptions. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 127–147. Springer, Cham (Aug 2022)
53. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Logrippo, L. (ed.) 10th ACM PODC. pp. 51–59. ACM (Aug 1991)
54. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT'99. LNCS, vol. 1592, pp. 223–238. Springer, Berlin, Heidelberg (May 1999)
55. Rachuri, R., Scholl, P.: Le mans: Dynamic and fluid MPC for dishonest majority. In: Dodis, Y., Shrimpton, T. (eds.) CRYPTO 2022, Part I. LNCS, vol. 13507, pp. 719–749. Springer, Cham (Aug 2022)

# Supplementary Material

## A Additional Preliminaries

In this section, we provide our additional technical preliminaries.

### A.1 Robust Threshold Secret Sharing

We recall the definition from [30]. A $(t, n)$-secure secret sharing scheme takes a secret $s$ as input and computes $n$ shares with the secrecy guarantee that any set of at most $t$ shares reveal nothing about $s$ while any set of $t+1$ correct shares can be used to efficiently reconstruct the secret. A robust secret sharing scheme additionally guarantees correct reconstruction from $n$ shares, even when an arbitrary set of $t$ out of the $n$ shares have been tampered. Formally,

**Definition 11 ($(t, n)$-Threshold Secret Sharing).** *A secret sharing scheme with secret domain $S$ and share domain $U$ is a pair of algorithms $(\mathsf{Sh}, \mathsf{Rec})$ called the sharing algorithm and the reconstruction algorithm, respectively.*

- ***Sharing:*** *The sharing algorithm $\mathsf{Sh}$ takes as input a secret $s \in S$ and randomness $\rho$ chosen uniformly from some domain $R$ and computes $(s_1, \ldots, s_n) \leftarrow \mathsf{Sh}(s, \rho)$, where $s_i \in U$ will be called the $i$-th share of the secret $s$. If $\rho$ is clear from the context we just write $(s_1, \ldots, s_n) \leftarrow \mathsf{Sh}(s)$. We denote by $[\![s]\!]$ a threshold sharing of $s$,*
- ***Reconstruction:*** *The reconstruction algorithm $\mathsf{Rec}$ takes as input the set of all shares and computes $\hat{s} \leftarrow \mathsf{Rec}(s_1, \ldots, s_n)$.*

*The $(t, n)$-secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$ should satisfy the following properties:*

- **Correctness:** *For all $\rho \leftarrow\!\!\$\ R$, and any $s \in S$, we have $\mathsf{Rec}(\mathsf{Sh}(s, \rho)) = s$.*
- **$t$-security:** *For any $s, s' \in S$, and for all subsets $\mathcal{I} \subset [n]$ such that $|\mathcal{I}| \leq t$, we have,*

$$\begin{aligned}\big\{\{s_i\}_{i\in\mathcal{I}} \mid \rho \leftarrow\!\!\$\ R, (s_1, \ldots, s_n) \leftarrow \mathsf{Sh}(s, \rho)\big\} \\ \equiv \big\{\{s_i\}_{i\in\mathcal{I}} \mid \rho \leftarrow\!\!\$\ R, (s_1, \ldots, s_n) \leftarrow \mathsf{Sh}(s', \rho)\big\}\end{aligned}$$

*The scheme $(\mathsf{Sh}, \mathsf{Rec})$ is said to be robust against $t$ corruptions if it additionally satisfies the following reconstruction property:*

- **$t$-robustness:** *For all $\rho \leftarrow\!\!\$\ R$, a secret $s \in S$, a set $\mathcal{I} \subset [n]$ of size at most $t$, and shares $\hat{s}_i \in U$, for $i \in \mathcal{I}$, we have,*

$$\mathsf{Rec}(\{s_i\}_{i\in[n]\setminus\mathcal{I}}, \{\hat{s}_i\}_{i\in\mathcal{I}}) = s, \text{ where } (s_1, \ldots, s_n) \leftarrow \mathsf{Sh}(s, \rho)$$

*Remark 12 (Threshold required for robustness).* As stated, $t$-robustness demands correct reconstruction for *every* choice of $t$ tampered shares with no failure probability, which requires $n \geq 3t + 1$: at $n = 2t + 1$ a degree-$t$ Shamir vector with $t$ errors lies within distance $t$ of several valid sharings, so no decoder can identify the secret. We therefore take $n_{\mathsf{Prep}} \geq 3t_{\mathsf{Prep}} + 1$ whenever $\Pi_{\mathsf{Prep}}$ is realized with plain Shamir sharing, which constrains preprocessing committees only and leaves the dishonest-majority computation committees untouched. Honest majority is recovered by instantiating $(\mathsf{Sh}, \mathsf{Rec})$ with a *statistical* robust or verifiable secret sharing scheme, which tolerates $t < n/2$ at the cost of a negligible failure probability, provided its verification material is distributed with the shares wherever $\mathsf{Rec}$ is later applied. In particular the MAC key shares $\alpha_i$ that $\mathcal{F}_{\mathsf{TE}}$ hands to preprocessing committees carry no such material as written.

### A.2 Linearly Homomorphic Public-Key Encryption Scheme

**Definition 13 (Linearly Homomorphic Public-Key Encryption).** *A linearly homomorphic public-key encryption scheme $\mathsf{Lin\text{-}PKE}$ consists of the following four PPT algorithms $(\mathsf{ParamsGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$, where:*

1. $\mathsf{pp} \leftarrow \mathsf{ParamsGen}(1^\kappa, q)$ *takes as input the security parameter* $\kappa$ *and some optional auxiliary information* $q$ *to generate public parameters* $\mathsf{pp}$ *for the scheme instantiation.*
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$ *takes as input the public parameters and outputs a public-secret key pair.*
3. $\mathsf{ct}_x \leftarrow \mathsf{Enc}(\mathsf{pk}, x; r)$ *takes as input some plain text* $x$ *and the public key* $\mathsf{pk}$ *and outputs a ciphertext* $\mathsf{ct}_x$ *using randomness* $r$*.*
4. $y := \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_y)$ *takes as input the secret key* $\mathsf{sk}$ *and some ciphertext* $\mathsf{ct}_y$ *and outputs the underlying plaintext* $y$*.*

**CPA Security:** *Any PKE scheme* $(\mathsf{ParamsGen}, \mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ *is* CPA-secure *if for all PPT adversaries* $\mathcal{A}$*, the advantage*

$$\mathsf{Adv}^{\mathsf{cpa}}_{\mathsf{PKE},\mathcal{A}}(\kappa) := \left|\Pr[b = b'] - \tfrac{1}{2}\right|$$

*is negligible in* $\kappa$*, where the probability is taken over the following experiment* $\mathsf{Exp}^{\mathsf{cpa}}_{\mathsf{PKE},\mathcal{A}}(\kappa)$*:*

1. $\mathsf{pp} \leftarrow \mathsf{ParamsGen}(1^\kappa, q)$.
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$.
3. $\mathcal{A}(\mathsf{pk})$ *outputs two plaintexts* $(m_0, m_1)$ *of equal length.*
4. $b \leftarrow\$ \{0,1\}$ *and* $\mathsf{ct}^\star \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
5. $\mathcal{A}(\mathsf{ct}^\star)$ *outputs a bit* $b'$*.*

*Modified IND-CPA Game.* Let $\mathcal{M}$ denote the distribution of secrets sampled by $\mathcal{F}_{\mathsf{Prep}}$. The modified IND-CPA experiment between a challenger $\mathcal{C}$ and a distinguisher $\mathcal{D}$ proceeds as follows:

1. $\mathsf{pp} \leftarrow \mathsf{ParamsGen}(1^\kappa, q)$.
2. $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{pp})$.
3. $\mathcal{D}(\mathsf{pk})$ outputs a message $m_0$, $\mathcal{C}$ samples $m_1 \leftarrow \mathcal{M}$.
4. $\mathcal{C}$ samples $b \leftarrow \{0,1\}$ and returns $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
5. $\mathcal{D}$ outputs a bit $b'$.

It is immediate that standard IND-CPA security implies security in this modified experiment, since an adversary in the standard game can internally sample $m_1 \leftarrow \mathcal{M}$ and submit $(m_0, m_1)$ to its challenger.

### A.3 Zero Knowledge Argument

We start by defining the notion of a $\Sigma-$protocol, and then define Zero-Knowledge Argument based off that. We restate this as in [17].

***Σ*-Protocols:** A $\Sigma$-protocol is a three-move protocol between a prover $P$ and a verifier $V$ for a witness relation $\mathsf{R}$. The common input are the relation $\mathsf{R}$ and a statement $x$, and $P$ additionally knows a witness $w$ such that $(x, w) \in \mathsf{R}$. We can specify a $\Sigma$-protocol by a tuple of three PPT algorithms $(\mathsf{Commit}, \mathsf{Respond}, \mathsf{Verify})$ and a challenge space $C$:

1. $P$ computes $(\mathsf{stt}, \mathsf{com}) \leftarrow \mathsf{Commit}(x, w)$ and sends the "commitment" $\mathsf{com}$.
2. $V$ sends a uniformly sampled challenge $\mathsf{chl} \leftarrow_\$ [C]$ from a challenge set $[C]$.
3. $P$ sends a response $\mathsf{res} \leftarrow \mathsf{Respond}(x, w, \mathsf{stt}, \mathsf{chl})$ to the challenge.

**Definition 14 (Zero-Knowledge Arguments).** *A protocol* $\Pi_{\mathsf{NIZK}} = (P, V)$ *that has the shape of a* $\Sigma$*-protocol* $(\mathsf{Commit}, \mathsf{Respond}, \mathsf{Verify})$ *for a witness relation* $\mathsf{R}$ *with witness domain* $D$ *is:*

1. ***Computationally sound*** *if no malicious* $P^*$ *can make* $V$ *accept a false statement* $x$ *(i.e., there is no* $w$ *such that* $(x, w) \in \mathsf{R}$*) with probability non-negligible in* $\kappa$*,*
2. ***Statistically special honest-verifier zero-knowledge*** *(SHVZK) for* $D$ *if there exists an algorithm* $\mathsf{Sim}_{\mathsf{NIZK}}$ *that for a statement* $x$ *and a given challenge* $\mathsf{chl}$ *produces matching* $(\mathsf{com}, \mathsf{res})$*, that is distributed statistically close (in* $\sigma$*) to real protocol transcripts with* $\mathsf{chl}$ *of* $(P, V)$ *on input* $(x, w) \in \mathsf{R}$ *where* $w \in D$*.*
3. ***Complete*** *for* $D$ *if in an honest interaction between* $P$ *and* $V$ *for* $(x, w) \in \mathsf{R}$ *with* $w \in D$*,* $V$ *always accepts.*

Notice that we define interactive notions above, and as in [17], one could make them non-interactive using the random oracle model and the Fischlin-transform [41] which will have the required properties.

**A $\Sigma$-Protocol for $\mathsf{R}_{\mathsf{Scal}}$:** The relation $\mathsf{R}_{\mathsf{Scal}}$ is decided using only $\mathsf{Enc}$ and the operations $\boxplus, \boxdot$ of $\mathsf{Lin\text{-}PKE}$, and so requires no structure beyond the additive homomorphism already assumed. To prove $\mathsf{ct}' = (\mu \boxdot \mathsf{ct}) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; r)$ for public $\mathsf{ct}, \mathsf{ct}'$, the prover samples $\tilde{\mu}, \tilde{r}$ and sends $\mathsf{com} := (\tilde{\mu} \boxdot \mathsf{ct}) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; \tilde{r})$; on challenge $e$ it responds with $z := \tilde{\mu} + e\mu$ and $\zeta := \tilde{r} + er$; the verifier accepts if

$$\mathsf{com} \boxplus (e \boxdot \mathsf{ct}') \;=\; (z \boxdot \mathsf{ct}) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; \zeta).$$

Special soundness and honest-verifier zero knowledge follow as usual, and the non-interactive argument is obtained as for $\mathsf{R}_{\mathsf{Enc}}$. Neither the statement nor the witness mentions the plaintext of $\mathsf{ct}$, so a prover holding only the scalar $\mu$ can produce the argument; this is what allows a checkpoint party to prove that it applied its own share of the blinding factor to the encrypted check value, which it cannot decrypt. When $\mathsf{Lin\text{-}PKE}$ is instantiated over a group of unknown order, as with Paillier [54] or class groups [16, 23], extraction requires the usual care over the challenge space, exactly as for $\mathsf{R}_{\mathsf{Enc}}$.

### A.4 Coin Tossing Protocol $\Pi_{\mathsf{CoinToss}}$

This protocol enables two checkpoint committees to jointly produce a public ciphertext $\mathsf{ct}_r = \mathsf{Enc}(\mathsf{pk}, r)$ encrypting a uniformly random field element $r \in \mathbb{F}_p$ without any single party learning $r$. Each party $P_i$ in the first committee independently samples $r_i \in_R \mathbb{F}_p$, computes $\mathsf{ct}_{r_i} = \mathsf{Enc}(\mathsf{pk}, r_i)$, and attaches a NIZK proof of correct encryption $\pi^{\mathsf{Enc}}_{r_i}$. The second committee verifies each proof, discards any malformed ciphertext, and homomorphically aggregates the remaining contributions: $\mathsf{ct}_r := \boxplus_i \mathsf{ct}_{r_i}$, encrypting $r = \sum_i r_i$. Bias resistance holds as long as at least one honest party contributes to the sum.

Protocol $\Pi_{\mathsf{CoinToss}}$ is invoked during setup to produce $\mathsf{ct}_\alpha$ and the key polynomial coefficients for threshold distribution, and at each era boundary to generate fresh preprocessing key material and the $2K(n_{\mathsf{Comp}} - 1)$ per-wire masking ciphertexts for state expansion.

**Public Parameters:** Initialising Checkpoint Committees $\{\mathcal{C}_1^{\mathsf{Check}}, \mathcal{C}_2^{\mathsf{Check}}\}$.

**Committee $\mathcal{C}_1^{\mathsf{Check}}$:** Each party $P_i \in \mathcal{C}_1^{\mathsf{Check}}$ does the following:

1. Sample a random field element $r_i$ from $\mathbb{F}_p$.
2. Compute $\mathsf{ct}_{r_i} = \mathsf{Encrypt}(\mathsf{pk}, r_i; \rho_{r_i})$ and $\pi^{\mathsf{Enc}}_{r_i} \leftarrow \mathsf{Prove}((\mathsf{pk}, \mathsf{ct}_{r_i}), (r_i, \rho_{r_i}))$.
3. Broadcast $\mathsf{ct}_{r_i}, \pi^{\mathsf{Enc}}_{r_i}$ to all parties in $\mathcal{C}_2^{\mathsf{Check}}$.

**Committee $\mathcal{C}_2^{\mathsf{Check}}$:** Each party $P_i \in \mathcal{C}_2^{\mathsf{Check}}$ does the following:

1. Compute $\mathsf{Verify}((\mathsf{pk}, \mathsf{ct}_{r_j}), \pi^{\mathsf{Enc}}_{r_j})$ for all broadcasted ciphertexts by $P_j \in \mathcal{C}_1^{\mathsf{Check}}$.
2. Output $\mathsf{ct}_r = \boxplus \mathsf{ct}_{r_j}$ for all $\mathsf{ct}_{r_j}$ broadcast by every $P_j \in \mathcal{C}_1^{\mathsf{Check}}$ that has passed the verification as stated above.

**Figure 8:** Protocol $\Pi_{\mathsf{CoinToss}}$

### A.5 Multiparty Computation Functionality

We recall the ideal functionality for arithmetic multi-party computation introduced in [29], which captures the online evaluation of an arithmetic circuit over a finite field. The functionality $\mathcal{F}_{\mathsf{AMPC}}$, shown in Figure 9, maintains an internal mapping from variable identifiers to field elements and supports input, addition, multiplication, and output operations. All arithmetic is performed modulo a public prime $p$, fixed during initialization. We state it in the private-output variant with guaranteed output delivery, as realized by an honest-majority protocol: each output reaches its recipient alone, and neither a withholding corrupt party nor the environment can block delivery. We realise some of our protocols in the $\mathcal{F}_{\mathsf{AMPC}}$-hybrid model.

### A.6 Commitments in the Random Oracle Model

We use a standard commitment functionality in the random oracle model. The functionality, denoted $\mathcal{F}_{\mathsf{Commit}}$ and shown in Figure 10, captures the ideal behavior of a commitment scheme: a committing party binds itself to a value while keeping it hidden until an explicit opening phase. The functionality

**Initialize:** On input $(sid, \mathsf{init}, p^k)$ from all honest parties, the functionality activates and stores the modulus p.
**Input:** On input $(sid, \mathsf{input}, P_i, \mathsf{varid}, x)$ from $P_i$ and $(sid, \mathsf{input}, P_i, \mathsf{varid}, ?)$ from all other honest parties, with $\mathsf{varid}$ a fresh identifier, the functionality stores $(\mathsf{varid}, x)$. If $P_i$ has not supplied $x$ by the end of its round, it stores $(\mathsf{varid}, 0)$.
**Add:** On command $(sid, \mathsf{add}, \mathsf{varid}_1, \mathsf{varid}_2, \mathsf{varid}_3)$ from all honest parties (if $\mathsf{varid}_1$, $\mathsf{varid}_2$ are present in memory and $\mathsf{varid}_3$ is not), the functionality retrieves $(\mathsf{varid}_1, x), (\mathsf{varid}_2, y)$ and stores $(\mathsf{varid}_3, x+y \mod p)$.
**Multiply:** On input $(sid, \mathsf{multiply}, \mathsf{varid}_1, \mathsf{varid}_2, \mathsf{varid}_3)$ from all honest parties (if $\mathsf{varid}_1$, $\mathsf{varid}_2$ are present in memory and $\mathsf{varid}_3$ is not), the functionality retrieves $(\mathsf{varid}_1, x), (\mathsf{varid}_2, y)$ and stores $(\mathsf{varid}_3, x \cdot y \mod p)$.
**Output:** On input $(sid, \mathsf{output}, P_i, \mathsf{varid})$ from all honest parties (if $\mathsf{varid}$ is present in memory), the functionality retrieves $(\mathsf{varid}, x)$ and outputs $x$ to $P_i$. If $P_i$ is corrupt, $x$ is also sent to $\mathcal{S}$. Delivery is not subject to approval by $\mathcal{S}$ or the environment.

**Figure 9:** Ideal Functionality $\mathcal{F}_{\mathsf{AMPC}}$ for Arithmetic MPC

**Commit:** Upon receiving $(\mathsf{Commit}, sid, v, i, \tau_v)$ from a party $P_i$ or the adversary $\mathcal{S}$ (if $P_i$ is corrupted), where $v$ is in a specific domain or $\perp$, it stores $(v, i, \tau_v)$ on a list and outputs $(i, \tau_v)$ to all players and the adversary.
**Open:** Upon receiving $(\mathsf{Open}, i, \tau_v)$ from a party $P_i$ or the adversary $\mathcal{S}$ (if $P_i$ is corrupted), the functionality outputs $(v, i, \tau_v)$ to all players and the adversary. If $(\mathsf{NoOpen}, i, \tau_v)$ is given by the adversary and $P_i$ is corrupt, then functionality outputs $(\perp, i, \tau_v)$ to all players.

**Figure 10:** Ideal Functionality $\mathcal{F}_{\mathsf{Commit}}$ for Commitments

**Commit:**
1. To commit to $v$, $P_i$ sets $x \leftarrow v \| r$, where $r$ is chosen uniformly at random in a determined domain and queries the random oracle $\mathcal{H}$ to get $c \leftarrow \mathcal{H}(x)$.
2. $P_i$ broadcasts $(c, i, \tau_v)$ where $\tau_v$ is a handle for the commitment.

**Open:**
1. In order to open a commitment $(c, i, \tau_v)$, where $c := \mathcal{H}(v \| r)$, $P_i$ broadcasts $(x = v \| r, i, \tau_v)$.
2. All players check if $H(x) = c$. They accept if and only if the check passes.

**Figure 11:** Protocol $\Pi_{\mathsf{Commit}}$ for Commitments

allows corrupted parties to refuse opening a commitment, while ensuring that honest commitments are binding and correctly opened.

We reuse both the ideal functionality and the corresponding realization from [28]. The protocol shown in Figure 11 realizes $\mathcal{F}_{\mathsf{Commit}}$ in the random oracle model under standard assumptions. The protocol is perfectly binding and computationally hiding in the random oracle model, assuming the random oracle behaves as a truly random function.

### A.7 Identity Based Encryption with Threshold ID Key Generation

In one of the optimizations for the preprocessing phase, we require a preprocessing committee to transfer correlated randomness to computation committees that are not yet online. In order to do so, we follow the approach of [18], leveraging a public bulletin board and an identity based encryption (IBE) scheme with threshold ID key generation. In such a scheme, the master secret and public key pair is generated by means of a distributed key generation (DKG) protocol and a the secret key for a specific ID is generated via a threshold protocol executed by parties holding master secret key shares.

**Identity Based Encryption** We recall the definition of an identity-based encryption (IBE) scheme [15] before defining an ideal functionality capturing IBE with threshold ID key generation.

**IBE Scheme.** An IBE scheme $\mathsf{IBE}$ consists of the following algorithms:

$\mathsf{Setup}(1^\kappa)$**.** The setup algorithm takes as input a security parameter $\kappa$ and returns a master key $\mathsf{msk}$ together with some publicly known system parameters $\mathsf{pp}_{\mathsf{IBE}}$ including a master public key $\mathsf{mpk}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. We assume that all algorithms takes $\mathsf{pp}_{\mathsf{IBE}}$ as input implicitly.

$\mathsf{IDKeyGen}(\mathsf{msk}, \mathsf{id})$**.** The identity key-generation algorithm takes as input $\mathsf{msk}$ and an identity $\mathsf{id} \in \{0,1\}^*$, and returns a decryption key $\mathsf{sk}_{\mathsf{id}}$ for $\mathsf{id}$.

$\mathsf{Enc}_{\mathsf{IBE}}(\mathsf{id}, m)$**.** The encryption algorithm takes as input an identity string $\mathsf{id} \in \{0,1\}^*$ and $m \in \mathcal{M}$. It returns a ciphertext $\mathsf{ct} \in \mathcal{C}$.

$\mathsf{Dec}_{\mathsf{IBE}}(\mathsf{ct}, \mathsf{sk}_{\mathsf{id}})$**.** The decryption algorithm takes as input $\mathsf{ct} \in \mathcal{C}$ and a decryption key $\mathsf{sk}_{\mathsf{id}}$. It returns $m \in \mathcal{M}$.

**Correctness.** An IBE scheme $\mathsf{IBE}$ should satisfy the standard correctness property, namely for $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{IDKeyGen}(\mathsf{msk}, \mathsf{id})$ and for any $m \in \mathcal{M}$, we must have:

$$\mathsf{Dec}_{\mathsf{IBE}}(\mathsf{Enc}_{\mathsf{IBE}}(\mathsf{id}, m), \mathsf{sk}_{\mathsf{id}}) = m.$$

where $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$

**Adaptive-identity security [15].** After the challenger runs the setup algorithm, the adversary has access to an oracle $\mathcal{O}_{\mathsf{msk}}$ that on input any $\mathsf{id}$, returns $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{IDKeyGen}(\mathsf{msk}, \mathsf{id})$. $\mathsf{Adv}$ may query the oracle on arbitrary identities of its choice even before selecting the messages $m_0, m_1$. More formally, we say that $\mathsf{IBE}$ is secure if any PPT adversary $\mathsf{Adv}$ has only negligibly greater than $1/2$ probability of correctly guessing the bit $b$ in the following game:

1. The challenger runs $\mathsf{Setup}$ and outputs $\mathsf{pp}_{\mathsf{IBE}}$ to $\mathsf{Adv}$.
2. $\mathsf{Adv}$ may query the oracle $\mathcal{O}_{\mathsf{msk}}$ that on any input $\mathsf{id}$ returns $\mathsf{sk}_{\mathsf{id}}$.
3. $\mathsf{Adv}$ outputs a target identity $\mathsf{id}^*$ and two equal-size messages $m_0, m_1 \in \mathcal{M}$.
4. The challenger selects a random bit $b$ and outputs $\mathsf{ct}^* \leftarrow \mathsf{Enc}_{\mathsf{IBE}}(\mathsf{id}^*, m_b)$ to $\mathsf{Adv}$.
5. $\mathsf{Adv}$ may continue to query $\mathcal{O}_{\mathsf{msk}}$ on any input $\mathsf{id} \neq \mathsf{id}^*$.
6. $\mathsf{Adv}$ outputs $b'$.

**Ideal Functionality** We capture an IBE scheme with threshold ID key generation in $\mathcal{F}_{\mathsf{IBE}}$ presented in Figure 12. This functionality takes an approach similar to that of $\mathcal{F}_{\mathsf{TE}}$, executing the setup and key generation algorithms internally and storing the master secret key $\mathsf{msk}$, while public parameters are output to all committees. Later on, when activated by sufficiently many parties, it uses $\mathsf{msk}$ to generate an set of ID keys, which are each output to a set of specific parties.

**Public Parameters:** Committees $\{\mathcal{C}_1, \cdots, \mathcal{C}_m\}$, Initialising Committees $\mathcal{C}_{\mathsf{init1}}$, $\mathcal{C}_{\mathsf{init2}}$.
**Init:** Upon receiving $(\mathsf{init}, sid, 1^\kappa)$ from all parties in $\mathcal{C}_{\mathsf{init}}$, $\mathcal{F}_{\mathsf{IBE}}$ generates $\mathsf{msk}, \mathsf{pp}_{\mathsf{IBE}} \leftarrow \mathsf{Setup}(1^\kappa)$. It stores $\mathsf{msk}, \mathsf{pp}_{\mathsf{IBE}}$ and sends $\mathsf{pp}_{\mathsf{IBE}}$ to all parties in $\mathcal{C}_1, \cdots, \mathcal{C}_m$. $\mathcal{F}_{\mathsf{IBE}}$ ensures that this method is called only once, and before other calls.
**IDKeyGen:** Upon receiving $(\mathsf{IDKeyGen}, sid, \mathsf{id}, \mathcal{P}_{\mathcal{R}})$ from all honest parties in $\mathcal{C}_i$ where $\mathsf{id}$ is a set of $\ell$ IDs and $\mathcal{P}_{\mathcal{R}}$ is a set of $\ell$ sets of receiver parties $\mathcal{P}_{\mathcal{R}}[k]$ for $k = 1, \ldots, \ell$, $\mathcal{F}_{\mathsf{IBE}}$ computes $\mathsf{sk}_{\mathsf{id}} := \mathsf{IDKeyGen}(\mathsf{msk}, \mathsf{id}_k)$ for all $\mathsf{id}_k \in \mathsf{id_k}$, and outputs $\mathsf{sk}_{\mathsf{id}_k}$ to all parties in $\mathcal{P}_{\mathcal{R}}[k]$. Ignore all future messages from parties in $\mathcal{C}_i$.

**Figure 12:** Functionality $\mathcal{F}_{\mathsf{IBE}}$

**Realizing $\mathcal{F}_{\mathsf{IBE}}$** Notice that for IBE schemes such as the seminal Boneh-Franklin construction [15], the master secret and public key pair corresponds to an ElGamal key pair. Hence, this key pair can be generated via a suitable UC-secure DKG protocol compatible with ElGamal, *e.g.* [21]). Similarly, the ID key generation is can be achieved via a threshold decryption protocol compatible with ElGamal, as it is achieved by computing $H(ID)^{\mathsf{msk}}$ for a public group element $H(ID)$ from master secret key shares $\mathsf{msk}_i$. Such a protocol can be constructed by using a UC NIZK to prove that partial keys $H(ID)^{\mathsf{msk}_i}$ have been correctly computed from threshold shares $\mathsf{msk}_i$ and then interpolating $H(ID)^{\mathsf{msk}}$ in the exponent. Hence, this functionality can be efficiently realized from existing protocols. In general, such this functionality can be realized from any IBE scheme by using a suitable UC-secure DKG protocol to generate the master key pair in such a way that a committee obtains $\mathsf{msk}$ pairs, along with an UC-secure protocol for generating secret keys for a publicly know ID.

### A.8 Public Bulletin Board

We will employ an authenticated public bulletin board in our optimizations for efficiently supporting computation committees who join after preprocessing and public verifiability. We describe an ideal functionality $\mathcal{F}_{\mathsf{BB}}$ capturing such a bulletin board in Figure 13. Notice that such a functionality can be realized by combining a PKI and a UC-secure digital signature scheme with a standard (non-authenticated) UC-secure bulletin board, *e.g.* a UC-secure blockchain [5, 6]. We adopt a simplified functionality for the sake of clarity.

$\mathcal{F}_{\mathsf{BB}}$ interacts with a set of committees $\{\mathcal{C}_1, \cdots, \mathcal{C}_m\}$ and keeps a counter $c$ initially set to 0, proceeding as follows:

**Write:** Upon receiving $(\mathsf{Write}, sid, m)$ from $P_i \in \mathcal{C}_j$, store the message $(c, P_i, m)$ and increment $c$.
**Read:** Upon receiving $(\mathsf{Read}, sid)$ from $P_i \in \mathcal{C}_j$, return all messages $(\cdot, \cdot, m)$ that are stored.

**Figure 13:** Functionality $\mathcal{F}_{\mathsf{BB}}$

## B Proof for Theorem 1

We show that the protocol presented in Figures 4,5, 6, 7 realizes $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ in Figure 2 in the presence of a malicious, static, rushing adversary $\mathcal{A}$. Each role of an era is instantiated by the next committee drawn from the corresponding pool; we let $\mathcal{I}^{\mathsf{Comp}}_{i,j}, \mathcal{I}^{\mathsf{Prep}}_{i}, \mathcal{I}^{\mathsf{Check}}_{i,k}$ denote the corrupted subsets of the committees filling roles $\mathcal{R}^{\mathsf{Comp}}_{i,j}, \mathcal{R}^{\mathsf{Prep}}_{i}, \mathcal{R}^{\mathsf{Check}}_{i,k}$. For any era $i \in [l]$, the static corruption bounds are:

- Preprocessing: $|\mathcal{I}^{\mathsf{Prep}}_{i}| < \frac{|\mathcal{R}^{\mathsf{Prep}}_{i}|}{2}$.
- Checkpoint: $|\mathcal{I}^{\mathsf{Check}}_{i,k}| < \frac{|\mathcal{R}^{\mathsf{Check}}_{i,k}|}{2}$ for each $k \in [4]$.
- Computation: $|\mathcal{I}^{\mathsf{Comp}}_{i,j}| < |\mathcal{R}^{\mathsf{Comp}}_{i,j}|$ for each $j \in [m]$.

The proof follows from a PPT simulation algorithm $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ that we define for $\Pi_{\mathsf{Era\text{-}MPC}}$, where $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ emulates the ideal functionalities $\mathcal{F}_{\mathsf{TE}}$ (Figure 1), $\mathcal{F}_{\mathsf{Commit}}$ (Figure 10) and $\mathcal{F}_{\mathsf{Prep}}$ (Figure 3) along with the messages the adversary expects from them. The simulator, detailed in Figures 14,15,16,17,18, also uses the SHVZK simulator $\mathsf{Sim}_{\mathsf{NIZK}}$ for $\Pi_{\mathsf{NIZK}}$ in a black-box way. Throughout, $\mathsf{CompCtr}, \mathsf{PrepCtr}, \mathsf{CheckCtr}$ denote the pool counters of Section 3.3, which determine the concrete committees instantiating each role.

**Setup Phase:** $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ does the following:

1. While emulating $\mathcal{F}_{\mathsf{TE}}$:
   (a) Receive $(\mathsf{init}, sid, 1^\kappa)$ from $\mathcal{A}$ on behalf of all parties $P_k \in \mathcal{I}^{\mathsf{Check}}_{\mathsf{init0}}$. Compute $\mathsf{pp}_{\mathsf{TE}} \leftarrow \mathsf{ParamsGen}(1^\kappa, q)$ and send it to $\mathcal{A}$.
   (b) Receive $(\mathsf{KGen}, sid)$ from $\mathcal{A}$ on behalf of all parties $P_k \in \mathcal{I}^{\mathsf{Check}}_{\mathsf{init1}}$. Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KGen}(\mathsf{pp}_{\mathsf{TE}})$. Send $\mathsf{pk}_{\mathsf{TE}}$ to $\mathcal{A}$.
2. For the $i$'th instance of $\Pi_{\mathsf{CoinToss}}$ starting in $\mathcal{C}^{\mathsf{Check}}_{\mathsf{init2}}$, $i \in [0, t_{\mathsf{Prep}}]$:
   (a) On behalf of all honest parties $P_j \in \mathcal{C}^{\mathsf{Check}}_{\mathsf{init2}} \setminus \mathcal{I}^{\mathsf{Check}}_{\mathsf{init2}}$, sample $f_{i,j}$ at random, compute $\mathsf{ct}_{f_{i,j}}, \tilde{\pi}^{\mathsf{Enc}}_{f_{i,j}}$ using $\mathsf{Enc}$ and $\mathsf{Sim}_{\mathsf{NIZK}}$ respectively and send to $\mathcal{A}$.
   (b) Receive on behalf of all corrupt parties $P_j \in \mathcal{I}^{\mathsf{Check}}_{\mathsf{init2}}$, $\mathsf{ct}_{f_{i,j}}, \pi^{\mathsf{Enc}}_{f_{i,j}}$ from $\mathcal{A}$. Run $\mathsf{Verify}((\mathsf{pk}, \mathsf{ct}_{f_{i,j}}), \pi^{\mathsf{Enc}}_{f_{i,j}})$ and discard anything that fails.
   (c) Use $\mathsf{sk}$ to decrypt and set $f_i := \sum_j f_{i,j}$ for the ones with valid proofs.
   (d) Compute ciphertexts $\mathsf{ct}_{f_i}$ and set $\alpha := f_0$. Set $f(x) = \alpha + \sum_{l=1}^{t_{\mathsf{Prep}}} f_l x^l$.
3. For all $k \in [n_{\mathsf{Prep}}]$, compute $\alpha_k = f(k)$. For every $P_j \in \mathcal{I}^{\mathsf{Prep}}_0$, send $\alpha_j$ to $\mathcal{A}$.
4. Emulate the **PrepInput** phase of $\mathcal{F}_{\mathsf{Prep}}$ using $\alpha$ to broadcast $\mathsf{ct}_{r_j}, \mathsf{ct}_{\gamma(r_j)}$ for all $P_j \in \mathcal{C}_0$ to $\mathcal{C}^{\mathsf{Check}}_{\mathsf{init4}}$.

**Figure 14:** Simulator $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ for $\Pi_{\mathsf{Era\text{-}MPC}}$ - Setup Phase

We argue that the view of $\mathcal{A}$ in the ideal-world execution with $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ is computationally indistinguishable from the real execution of $\Pi_{\mathsf{Era\text{-}MPC}}$ through the following sequence of hybrids.

**Input Phase:** $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ does the following:

1. For $j \in [n_{\mathsf{in}}], k \in [n_{\mathsf{Comp}} - 1]$, do the following for $\Pi_{\mathsf{CoinToss}}$ starting in $\mathcal{C}^{\mathsf{Check}}_{\mathsf{init4}}$:
   (a) On behalf of all honest parties $P_l \in \mathcal{C}^{\mathsf{Check}}_{\mathsf{init4}} \setminus \mathcal{I}^{\mathsf{Check}}_{\mathsf{init4}}$, sample $r_{j,k,l}$ at random, compute $\mathsf{ct}_{r_{j,k,l}}, \tilde{\pi}^{\mathsf{Enc}}_{\mathsf{ct}_{r_{j,k,l}}}$ using $\mathsf{Enc}$ and $\mathsf{Sim}_{\mathsf{NIZK}}$ respectively and send to $\mathcal{A}$.
   (b) Receive on behalf of all corrupt parties $P_l \in \mathcal{I}^{\mathsf{Check}}_{\mathsf{init4}}$, $\mathsf{ct}_{r_{j,k,l}}, \pi^{\mathsf{Enc}}_{\mathsf{ct}_{r_{j,k,l}}}$ from $\mathcal{A}$. Run $\mathsf{Verify}(\mathsf{ct}_{r_{j,k,l}}, \pi^{\mathsf{Enc}}_{\mathsf{ct}_{r_{j,k,l}}})$ and discard anything that fails.
   (c) Use $\mathsf{sk}$ to decrypt and set $r_{j,k} := \sum_l r_{j,k,l}$ for the ones with valid proofs.
   (d) Compute ciphertexts $\mathsf{ct}_{r_{j,k}}$. Repeat all steps for computing $\mathsf{ct}_{\gamma(r_j)_k}$.
2. Send $r_i$ corresponding to all corrupt parties $P_i \in \mathcal{I}_0$ on behalf of $\mathcal{F}_{\mathsf{TE}}$.
3. For all honest parties $P_i \in \mathcal{C}_0 \setminus \mathcal{I}_0$, sample $\epsilon_i$ and send to $\mathcal{A}$. Set $\tilde{x}_i := \epsilon_i + r_i$.
4. On behalf of all corrupt parties $P_i \in \mathcal{I}_0$, receive $\epsilon_i$ from $\mathcal{A}$. Extract $\tilde{x}_i = \epsilon_i + r_i$.
5. Send to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ on behalf of all corrupt parties $P_i \in \mathcal{I}_0$, $(sid, \mathsf{input}, P_i, \tilde{x}_i)$. For inputs not received, send default inputs.
6. For $j \in [n_{\mathsf{in}}]$, do the following:
   (a) For $k \in [n_{\mathsf{Comp}} - 1]$, $\mathsf{ct}_{[\tilde{x}_j]_k} := \mathsf{ct}_{r_{j,k}}, \mathsf{ct}_{[\gamma(\tilde{x}_j)_k]} := \mathsf{ct}_{\gamma(r_j)_k}$.
   (b) Set $\mathsf{ct}_{[\tilde{x}_j]_{n_{\mathsf{Comp}}}} := \mathsf{ct}_{\epsilon_j} \boxplus \mathsf{ct}_{r_j} \boxplus (-1 \boxdot \boxplus_{k=1}^{n_{\mathsf{Comp}}-1} \mathsf{ct}_{[\tilde{x}_j]_k})$.
   (c) Set $\mathsf{ct}_{[\gamma(\tilde{x}_j)]_{n_{\mathsf{Comp}}}} := (\epsilon_j \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{ct}_{\gamma(r_j)} \boxplus (-1 \boxdot \boxplus_{k=1}^{n_{\mathsf{Comp}}-1} \mathsf{ct}_{[\gamma(\tilde{x}_j)]_k})$.
7. For the $i$'th instance of $\Pi_{\mathsf{CoinToss}}$ starting in $\mathcal{C}^{\mathsf{Check}}_{0,1}$, $i \in [t_{\mathsf{Prep}}]$:
   (a) On behalf of all honest parties $P_j \in \mathcal{C}^{\mathsf{Check}}_{0,1} \setminus \mathcal{I}^{\mathsf{Check}}_{0,1}$, sample $f_{i,j}$ at random, compute $\mathsf{ct}_{f_{i,j}}, \tilde{\pi}^{\mathsf{Enc}}_{f_{i,j}}$ using $\mathsf{Enc}$ and $\mathsf{Sim}_{\mathsf{NIZK}}$ respectively and send to $\mathcal{A}$.
   (b) Receive on behalf of all corrupt parties $P_j \in \mathcal{I}^{\mathsf{Check}}_{0,1}$, $\mathsf{ct}_{f_{i,j}}, \pi^{\mathsf{Enc}}_{f_{i,j}}$ from $\mathcal{A}$. Run $\mathsf{Verify}((\mathsf{pk}, \mathsf{ct}_{f_{i,j}}), \pi^{\mathsf{Enc}}_{f_{i,j}})$ and discard anything that fails.
   (c) Use $\mathsf{sk}$ to decrypt and set $f_i := \sum_j f_{i,j}$ for the ones with valid proofs.
   (d) Compute ciphertexts $\mathsf{ct}_{f_i}$ and set $f(x) = \alpha + \sum_{l=1}^{t_{\mathsf{Prep}}} f_l x^l$.
8. For all $k \in [n_{\mathsf{Prep}}]$, compute $\alpha_k = f(k)$. For every $P_j \in \mathcal{I}^{\mathsf{Prep}}_1$, send $\alpha_j$ to $\mathcal{A}$.
9. Emulate the **PrepCompute** phase of $\mathcal{F}_{\mathsf{Prep}}$ on input $(\mathsf{PrepCompute}, sid, 1, \mathsf{CompCtr}, \mathsf{CheckCtr}, \{\mathsf{int}_{1,j}\}_{j\in[m]}, \ldots)$ by running the code as is; for $j \in [m], k \in [\mathsf{int}_{1,j}]$, output corrupt-party shares of $\langle a_{j,k}\rangle, \langle b_{j,k}\rangle, \langle c_{j,k}\rangle, \langle 1\rangle^j, [\beta_k^{1,j}]^j, [\eta_k^{1,j}]^j$ (and $[\beta_k^{1,j}]^{j+1}, [\eta_k^{1,j}]^{j+1}$ for $j < m$) together with $\mathsf{ct}_R, \mathsf{ct}_{\gamma(R)}$ to $\mathcal{A}$. Set $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}} := \{\{\mathsf{ct}_{[x_i]_j}, \mathsf{ct}_{[\gamma(x_i)_j]}\}_{i\in[n_{\mathsf{in}}]}\}_{j\in[n_{\mathsf{Comp}}]}$ and send it to $\mathcal{A}$. Also send $\{[x_i]_j, [\gamma(x_i)_j]\}_{i\in[n_{\mathsf{in}}]}$ for every $P_j \in \mathcal{I}^{\mathsf{Comp}}_{1,1}$ to $\mathcal{A}$.

**Figure 15:** Simulator $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ for $\Pi_{\mathsf{Era\text{-}MPC}}$ - Input Phase

$\mathsf{Hyb}_0$. The real-world execution of $\Pi_{\mathsf{Era\text{-}MPC}}$ in the $(\mathcal{F}_{\mathsf{Prep}}, \mathcal{F}_{\mathsf{TE}}, \mathcal{F}_{\mathsf{Commit}})$-hybrid model.

$\mathsf{Hyb}_1$. Identical to $\mathsf{Hyb}_0$, except that all arguments generated by honest parties, for $\mathsf{R}_{\mathsf{Enc}}$ in $\Pi_{\mathsf{CoinToss}}$ and **CheckOpen**, for $\mathsf{R}_{\mathsf{Enc}}$ in **Commit-and-Encrypt**, and for $\mathsf{R}_{\mathsf{Scal}}$ in the blinding step at $\mathcal{R}^{\mathsf{Check}}_{i,2}$, are produced using $\mathsf{Sim}_{\mathsf{NIZK}}$ instead of the honest prover. Indistinguishability follows from the SHVZK property of $\Pi_{\mathsf{NIZK}}$.

$\mathsf{Hyb}_2$. Identical to $\mathsf{Hyb}_1$, except that every ciphertext broadcast on behalf of honest parties — the CheckOpen tag ciphertexts $\mathsf{ct}_{\gamma_k^j}$, the Commit-and-Encrypt share ciphertexts $(\mathsf{ct}_{y_{k,j}}, \mathsf{ct}_{\gamma(y_k)_j})$, the blinded check ciphertexts $\mathsf{ct}_{c,l}$ of $\mathcal{R}^{\mathsf{Check}}_{i,2}$, the compact state ciphertexts $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}}, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$, and the $\Pi_{\mathsf{CoinToss}}$ ciphertexts — is replaced by a fresh encryption of a fixed dummy plaintext consistent with the corrupt parties' decryptable shares. The plaintexts $\mathcal{A}$ also sees, namely the openings $\tilde{\epsilon}, \tilde{\rho}$, the masked handoff broadcasts, the shares $\mathcal{F}_{\mathsf{TE}}$ decrypts to corrupt recipients, and the released $y'$ and $r \cdot c$, are reproduced exactly as in $\mathsf{Hyb}_1$, and the replacements are chosen consistently with them.

What is preserved is the public aggregation *identities*, not the aggregate ciphertexts: each $\mathsf{ct}_{y_k}$ is still the $\boxplus$ of the broadcast share ciphertexts, and $\mathsf{ct}_\sigma, \mathsf{ct}_c, \mathsf{ct}_{r\cdot c}$ are still the prescribed combinations of them. The aggregates must be allowed to change, since with one honest computation party, which $t_{\mathsf{Comp}} < n_{\mathsf{Comp}}$ permits, that party's ciphertext is determined by the aggregate and the corrupt contributions, so fixing the aggregate would leave no plaintext free to replace. The reduction justifies that change: corrupt contributions are unchanged, identities are recomputed as the protocol prescribes, and $y'$ and $r \cdot c$ are held at their $\mathsf{Hyb}_1$ values by the programming of $\mathcal{F}_{\mathsf{TE}}$ in Figures 17,18. Conditioned on the fixed plaintexts, the remaining view is the replaced ciphertexts and homomorphic functions of them, so a distinguisher yields an IND-CPA distinguisher for $\mathsf{Lin\text{-}PKE}$.

**Computation Phase:** For $i \in [l]$ and $j \in [m]$, executed sequentially:

1. Send $(sid, \mathsf{Eval}, i, j)$ to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ on behalf of all corrupt parties in $\mathcal{R}^{\mathsf{Comp}}_{i,j}$.
2. **Evaluation of sub-circuit $\mathbf{C}_{i,j}$:** The simulator executes the circuit using the shares of the dummy values $\tilde{x}_i$ computed during the Input Phase.
   - **Addition Gates:** Update dummy shares: $\langle \tilde{z} \rangle = \langle \tilde{x} \rangle + \langle \tilde{y} \rangle$.
   - **Multiplication Gates:**
     (a) Using the triples generated in the $\mathcal{F}_{\mathsf{Prep}}$ emulation, $\mathsf{Sim}$ computes the dummy openings $\tilde{\epsilon} = \tilde{x} - a$ and $\tilde{\rho} = \tilde{y} - b$.
     (b) Reveal shares of honest parties in $\mathcal{R}^{\mathsf{Comp}}_{i,j}$ for openings $\tilde{\epsilon}, \tilde{\rho}$ to $\mathcal{A}$.
     (c) Receive shares of $\tilde{\epsilon}, \tilde{\rho}$ on behalf of corrupt parties in $\mathcal{R}^{\mathsf{Comp}}_{i,j}$. If $\mathcal{A}$ fails to provide them, send $(sid, \mathsf{EraAbort})$ to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$; otherwise record the offset by which each opening differs from the value prescribed by the corrupt shares and continue.
     (d) Compute the next dummy wire sharing: $\langle \tilde{z} \rangle = \langle c \rangle + \tilde{\epsilon} \cdot \langle b \rangle + \tilde{\rho} \cdot \langle a \rangle + \tilde{\epsilon} \cdot \tilde{\rho} \cdot \langle 1 \rangle$.
3. **CheckOpen:** Let $(o^j_1, \cdots, o^j_{t'})$ be the list of publicly opened values to $\mathcal{A}$ on behalf of the honest parties, and $\{[\gamma(o^j_1)], \cdots, [\gamma(o^j_{t'})]\}$ the sharings of their tags.
   (a) Receive $(\mathsf{Commit}, s_k)$ for corrupt parties on behalf of $\mathcal{F}_{\mathsf{Commit}}$.
   (b) Sample $s_k$ at random on behalf of all honest $P_k \in \mathcal{R}^{\mathsf{Comp}}_{i,j}$ and send it to $\mathcal{A}$.
   (c) If $\mathcal{A}$ sends $\mathsf{NoOpen}$, or fails to send $\mathsf{Open}$, send $(sid, \mathsf{EraAbort})$ to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$.
   (d) Set $s = s_1 \oplus \cdots \oplus s_{n_{\mathsf{Comp}}}$ and $\mathbf{r} = \mathcal{U}_s(p, t')$.
   (e) Compute $o^j = \sum r_l o^j_l$ and tag shares $\gamma^j_k = \sum r_l \gamma(o^j_l)_k$.
   (f) Compute $\mathsf{ct}_{\gamma^j_k}$ and $\pi^{\mathsf{Enc}}_{\gamma^j_k} \leftarrow \mathsf{Sim}_{\mathsf{NIZK}}(\mathsf{pk}, \mathsf{ct}_{\gamma^j_k})$. Broadcast $(o^j, \mathsf{ct}_{\gamma^j_k}, \pi^{\mathsf{Enc}}_{\gamma^j_k})$ to $\mathcal{R}^{\mathsf{Check}}_{i,1}$ for all honest $P_k \in \mathcal{R}^{\mathsf{Comp}}_{i,j} \setminus \mathcal{I}^{\mathsf{Comp}}_{i,j}$.
   (g) Receive $(o^j, \mathsf{ct}_{\gamma^j_k}, \pi^{\mathsf{Enc}}_{\gamma^j_k})$ from $\mathcal{A}$ for all corrupt parties $P_k \in \mathcal{I}^{\mathsf{Comp}}_{i,j}$. Use $\mathsf{sk}$ to extract the plaintext tags $\gamma^j_k$.
4. **StateReshare:**
   - If $j < m$: For each intermediate output wire $k$ of sub-circuit $\mathbf{C}_{i,j}$:
     (a) On behalf of each honest party $P_l \in \mathcal{R}^{\mathsf{Comp}}_{i,j} \setminus \mathcal{I}^{\mathsf{Comp}}_{i,j}$, broadcast $[\tilde{y}^{i,j}_k]_l + [\beta^{i,j}_k]^j_l$ and $[\gamma(\tilde{y}^{i,j}_k)]_l + [\eta^{i,j}_k]^j_l$ to $\mathcal{R}^{\mathsf{Comp}}_{i,j+1}$, sending both to $\mathcal{A}$.
     (b) Receive the two masked broadcasts of each corrupt party $P_u \in \mathcal{I}^{\mathsf{Comp}}_{i,j}$ from $\mathcal{A}$, and record the offsets $(\Delta_{u,k}, \Delta'_{u,k})$ by which they differ from the values prescribed by $P_u$'s shares.
     (c) For each honest $P_l \in \mathcal{R}^{\mathsf{Comp}}_{i,j+1} \setminus \mathcal{I}^{\mathsf{Comp}}_{i,j+1}$, set its shares of $\langle \tilde{y}'_k \rangle$ by subtracting $[\beta^{i,j}_k]^{j+1}_l$ and $[\eta^{i,j}_k]^{j+1}_l$ from the two values broadcast by $P_l \in \mathcal{R}^{\mathsf{Comp}}_{i,j}$.
   - If $j = m$: Refer to Figures 17,18 for the era-level state transfer simulation.

**Output Phase:** Once $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ sends $\mathsf{OutputSuccess}$ at the close of the final era $l$:

- Receive the ideal output $(sid, \mathsf{Output}, \mathbf{y}_{l,m})$ from $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$.
- On behalf of all honest parties, adjust their shares of the output wires so that the value reconstructed by $\mathcal{A}$ is exactly $y_k$ for each $k \in [\mathsf{int}_{l,m}]$, and program $\mathcal{F}_{\mathsf{TE}}$ to decrypt the output ciphertexts $\{\mathsf{ct}_{y_k}\}$ computed by $\mathcal{R}^{\mathsf{Check}}_{l,4}$ to $\mathcal{C}_{\mathsf{out}}$ accordingly.

**Figure 16:** Simulator $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ for $\Pi_{\mathsf{Era\text{-}MPC}}$ - **Evaluate Phase**

$\mathsf{Hyb}_3$. Identical to $\mathsf{Hyb}_2$, except that for every honest party $P_i$ the actual input $x_i$ is replaced by a programmed value $\tilde{x}_i = \epsilon_i + r_i$ for a uniformly chosen $\epsilon_i$; all subsequent honest values — shares, MAC tags, intermediate wires, and ciphertexts — are computed consistently with respect to $\tilde{x}_i$. In multiplication gates, the opened values $\tilde{\epsilon} = \tilde{x} - a$ and $\tilde{\rho} = \tilde{y} - b$ are uniformly distributed by the one-time-pad property of the Beaver triples from $\mathcal{F}_{\mathsf{Prep}}$.

*Independence of the handoff masks.* In the intra-era **StateReshare** step each honest party broadcasts the pair $\left([\tilde{y}^{i,j}_k]_l + [\beta^{i,j}_k]^j_l,\ [\gamma(\tilde{y}^{i,j}_k)]_l + [\eta^{i,j}_k]^j_l\right)$, where $\beta^{i,j}_k$ and $\eta^{i,j}_k$ are sampled independently of one another and of $\tilde{y}^{i,j}_k$ by $\mathcal{F}_{\mathsf{Prep}}$. Both broadcasts are therefore independent one-time pads, and the pair is distributed uniformly on $\mathbb{F}_p^2$. Independence of the two masks is what the argument rests on: had the tag been masked by $\alpha \cdot \beta^{i,j}_k$, as a single authenticated sharing $\left\langle \beta^{i,j}_k \right\rangle$ would give, the two broadcasts would sum to $(u + \beta, \alpha u + \alpha\beta)$ for a single unknown $\beta$, from which $\mathcal{A}$ recovers $u$ and $\alpha u$ and hence $\alpha$ itself. A further consequence is that secrecy here does not depend on who receives the values, so the handoff may be a broadcast rather than a point-to-point transfer; the matching sharings $[\beta^{i,j}_k]^{j+1}, [\eta^{i,j}_k]^{j+1}$ held by $\mathcal{R}^{\mathsf{Comp}}_{i,j+1}$ let each party of the same index strip its own pads locally. Hence the view in $\mathsf{Hyb}_3$ is distributed identically to that in $\mathsf{Hyb}_2$.

**Simulating for $\mathcal{R}^{\mathsf{Comp}}_{i,m}$ (Commit-and-Encrypt):**

1. On behalf of honest parties, compute $(\mathsf{ct}_{\tilde{y}_{k,j}}, \mathsf{ct}_{\gamma(\tilde{y}_k)_j})$ and $\mathsf{Sim}_{\mathsf{NIZK}}$ proofs using dummy shares $\langle \tilde{y}_{k,j} \rangle$ from the sub-circuit evaluation, and broadcast them to $\mathcal{A}$.
2. Receive ciphertexts and proofs from $\mathcal{A}$ for corrupt parties. Extract the adversary's shares $(y_{k,j}, \gamma(y_k)_j)$ using $\mathsf{sk}$. If NIZK proofs fail, send $(sid, \mathsf{EraAbort})$ to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ and send $(\mathsf{ct}_\alpha, \perp, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ to $\mathcal{A}$.
3. Receive $(\mathsf{Commit}, s_j)$ for corrupt parties on behalf of $\mathcal{F}_{\mathsf{Commit}}$.
4. Sample $s_j$ at random on behalf of all honest $P_j \in \mathcal{R}^{\mathsf{Comp}}_{i,m}$ and send it to $\mathcal{A}$.
5. If $\mathcal{A}$ sends $\mathsf{NoOpen}$, or fails to send $\mathsf{Open}$, send $(sid, \mathsf{EraAbort})$ to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$.
6. Set $s = s_1 \oplus \cdots \oplus s_{n_{\mathsf{Comp}}}$ and $\mathbf{r}' = \mathcal{U}_s(p, \mathsf{int}_{i,m})$, and broadcast $\mathbf{r}'$ to $\mathcal{R}^{\mathsf{Check}}_{i,1}$.

**Simulating for $\mathcal{R}^{\mathsf{Check}}_{i,1}$ (MAC and Share Aggregation):**

1. If any received proof fails to verify, or any expected message from $\mathcal{R}^{\mathsf{Comp}}_{i,m}$ is missing, broadcast $(\mathsf{ct}_\alpha, \perp, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ to $\mathcal{R}^{\mathsf{Check}}_{i,2}$ on behalf of the honest parties and stop.
2. Homomorphically aggregate the extracted adversary shares and the honest dummy shares into $\mathsf{ct}_{\tilde{y}_k}, \mathsf{ct}_{\gamma(\tilde{y}_k)}$, and set $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}} := \{\mathsf{ct}_{\tilde{y}_k}, \mathsf{ct}_{\gamma(\tilde{y}_k)}\}_{k \in [\mathsf{int}_{i,m}]}$.
3. Compute $\mathsf{ct}_{\tilde{y}+R}, \mathsf{ct}_{\gamma(\tilde{y}+R)}$ from $\mathbf{r}'$ and the pair $(R, \gamma(R))$, and $\mathsf{ct}_\sigma$ as in the protocol.
4. Program $\mathcal{F}_{\mathsf{TE}}$ to return $y' = \sum_k r'_k \tilde{y}_k + R$ to $\mathcal{R}^{\mathsf{Check}}_{i,2}$, the $\mathbf{r}'$-weighted reconstruction of the **dummy wire state** plus the mask $R$.

**Figure 17:** Simulator $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ for $\Pi_{\mathsf{Era\text{-}MPC}}$ - **State Transfer**

**Simulating for $\mathcal{R}^{\mathsf{Check}}_{i,2}$ (MAC Validation and Blinding):**

1. If $(\mathsf{ct}_\alpha, \perp, \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}})$ was received, forward it to $\mathcal{R}^{\mathsf{Check}}_{i,3}$ on behalf of the honest parties and stop.
2. Since $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ knows $\alpha$ and all extracted shares, calculate $c = (y' \cdot \alpha) + \sigma$.
3. On behalf of each honest $P_l \in \mathcal{R}^{\mathsf{Check}}_{i,2} \setminus \mathcal{I}^{\mathsf{Check}}_{i,2}$, sample $r_l \leftarrow\!\$\ \mathbb{F}_p$, set $\mathsf{ct}_{c,l} := (r_l \boxdot \mathsf{ct}_c) \boxplus \mathsf{Encrypt}(\mathsf{pk}, 0; s_l)$, compute $\tilde{\pi}^{\mathsf{Scal}}_l$ using $\mathsf{Sim}_{\mathsf{NIZK}}$, and broadcast $(\mathsf{ct}_{c,l}, \tilde{\pi}^{\mathsf{Scal}}_l)$ to $\mathcal{A}$.
4. Receive $(\mathsf{ct}_{c,l}, \pi^{\mathsf{Scal}}_l)$ from $\mathcal{A}$ for all corrupt $P_l \in \mathcal{I}^{\mathsf{Check}}_{i,2}$.

**Simulating for $\mathcal{R}^{\mathsf{Check}}_{i,3}$ (Aggregation):**

1. Simulate $t_{\mathsf{Prep}} + 2K_i(n_{\mathsf{Comp}} - 1)$ instances of $\Pi_{\mathsf{CoinToss}}$ with $\mathcal{A}$ as in Figure 15: $t_{\mathsf{Prep}}$ instances to produce the polynomial coefficient ciphertexts $\mathsf{ct}_{f_1}, \cdots, \mathsf{ct}_{f_{t_{\mathsf{Prep}}}}$, and $2K_i(n_{\mathsf{Comp}} - 1)$ instances to produce the per-wire masking ciphertext pairs $\{\mathsf{ct}_{r_{k,l}}, \mathsf{ct}_{\gamma(r_k)_l}\}_{k \in [K_i], l \in [n_{\mathsf{Comp}}-1]}$ for the state expansion at $\mathcal{R}^{\mathsf{Check}}_{i,4}$.
2. Run $\mathsf{Verify}((\mathsf{pk}, \mathsf{ct}_c, \mathsf{ct}_{c,l}), \pi^{\mathsf{Scal}}_l)$ for the ciphertexts received from $\mathcal{A}$, let $V$ be the indices that verify together with the honest indices, and form $\mathsf{ct}_{r \cdot c} := \boxplus_{l \in V} \mathsf{ct}_{c,l}$.
3. Program $\mathcal{F}_{\mathsf{TE}}$ to return to $\mathcal{R}^{\mathsf{Check}}_{i,4}$ the value $c' := 0$ if $c = 0$, and $c' \leftarrow\!\$\ \mathbb{F}_p$ otherwise.

**Simulating for $\mathcal{R}^{\mathsf{Check}}_{i,4}$ (State Update or Rollback):**

1. **Commit or roll back.** If $c' \neq 0$ or an abort occurred: send $(sid, \mathsf{EraAbort})$ to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ and set $\mathsf{ct}_{\mathsf{St}_{\mathsf{Cur}}} := \mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ (roll back). Otherwise ($c' = 0$): set the dummy state as the new $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ (commit).
2. **Final era.** If $i = l$ and $c' = 0$, the last era is committed: do not advance the counters, run the abort-check, select new committees, or emulate $\mathcal{F}_{\mathsf{Prep}}$; instead proceed to the **Output Phase** of Figure 16. Skip the remaining steps.
3. Advance the counters $\mathsf{CompCtr} \leftarrow \mathsf{CompCtr} + m$, $\mathsf{PrepCtr} \leftarrow \mathsf{PrepCtr} + 1$, $\mathsf{CheckCtr} \leftarrow \mathsf{CheckCtr} + 4$.
4. If the pools are exhausted (the three-counter abort condition of $\Pi_{\mathsf{StateTransfer}}$ trips), $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ outputs $\perp$ to $\mathcal{C}_{\mathsf{out}}$; deliver $\perp$ to all corrupt parties in $\mathcal{C}_{\mathsf{out}}$ and terminate.
5. Deliver the notification from $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ to all corrupt parties in the newly assigned block of committees: on rollback, $\mathsf{EraFail}$ tagged with the era $i$ being retried (filling $\mathcal{R}^{\mathsf{Comp}}_{i,1}, \ldots, \mathcal{R}^{\mathsf{Comp}}_{i,m}$); on commit, $\mathsf{EraSuccess}$ tagged with the just-committed era $i$ (filling $\mathcal{R}^{\mathsf{Comp}}_{i+1,1}, \ldots, \mathcal{R}^{\mathsf{Comp}}_{i+1,m}$).
6. Using the masking ciphertext pairs $\{\mathsf{ct}_{r_{k,l}}, \mathsf{ct}_{\gamma(r_k)_l}\}$ from $\mathcal{R}^{\mathsf{Check}}_{i,3}$, expand the chosen compact state into per-party SPDZ shares: for each wire $k \in [K_i]$, set $\mathsf{ct}_{[y_k]_l} := \mathsf{ct}_{r_{k,l}}$ and $\mathsf{ct}_{[\gamma(y_k)]_l} := \mathsf{ct}_{\gamma(r_k)_l}$ for $l \in [n_{\mathsf{Comp}} - 1]$, then derive the $n_{\mathsf{Comp}}$-th party's ciphertexts homomorphically.
7. Emulate $\mathcal{F}_{\mathsf{TE}}$ to deliver the shares $\alpha_j$ to $\mathcal{A}$ for corrupt parties $P_j \in \mathcal{R}^{\mathsf{Prep}}_{i+1}$.
8. Emulate the **PrepCompute** phase of $\mathcal{F}_{\mathsf{Prep}}$ for the next era by running the code as is, outputting corrupt-party shares of $\langle a_{j,k} \rangle, \langle b_{j,k} \rangle, \langle c_{j,k} \rangle, \langle 1 \rangle^j, [\beta_k^{i+1,j}]^j, [\eta_k^{i+1,j}]^j$ (and $[\beta_k^{i+1,j}]^{j+1}, [\eta_k^{i+1,j}]^{j+1}$ for $j < m$) and $\mathsf{ct}_R, \mathsf{ct}_{\gamma(R)}$ to $\mathcal{A}$. Send $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ to $\mathcal{A}$, and on behalf of $\mathcal{F}_{\mathsf{TE}}$ deliver the expanded next-era shares to $\mathcal{R}^{\mathsf{Comp}}_{i+1,1}$.

**Figure 18:** Simulator $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ for $\Pi_{\mathsf{Era\text{-}MPC}}$ - **State Transfer Continued**

*MAC soundness.* Since $\alpha$ is the constant term of a degree-$t_{\mathsf{Prep}}$ polynomial with honest-majority preprocessing ($n_{\mathsf{Prep}} > 2t_{\mathsf{Prep}}$), the adversary holds at most $t_{\mathsf{Prep}}$ evaluation shares $\alpha_k = f(k)$ and learns nothing about $\alpha = f(0)$. The invariant to establish is not that every deviation is rejected, since deviations that cancel or that leave the committed state correct are harmless and need not

abort, but that no *incorrect* state or opening is accepted. Every deviation available to a corrupt computation committee enters the era transcript additively: a forged tag share in **CheckOpen**, a mis-encrypted share in **Commit-and-Encrypt**, and the offsets $(\Delta_{u,k}, \Delta'_{u,k})$ injected by a corrupt broadcast in **StateReshare** all shift $c = \sum_j(-\alpha \cdot o^j + \sum_l \gamma_l^j)$ by a quantity fixed before $\alpha$ is used. Authenticating the public term of each multiplication by $\langle 1 \rangle$ is what keeps this equation closed, since an unauthenticated $\epsilon \cdot \rho$ would contribute to the value but not to the tag. Combined with the challenges $\mathbf{r}$ and $\mathbf{r}'$, an era whose committed state or openings differ from the correct ones therefore yields $c \neq 0$ except with probability at most $1/p$, while deviations whose net effect on the state is nil leave $c = 0$ and commit a correct state.

Two further points concern when the challenge is fixed and how the check is released. First, in **Commit-and-Encrypt** the share ciphertexts are broadcast before any seed is committed, so $\mathbf{r}'$ is fixed only after $\mathcal{A}$'s contributions are and have been decrypted by $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ using $\mathsf{sk}$; were the seed opened first, a corrupt committee could choose contributions cancelling under the known $\mathbf{r}'$. Second, $\mathcal{R}_{i,3}^{\mathsf{Check}}$ decrypts $r \cdot c$, not $c$. Every honest party of $\mathcal{R}_{i,2}^{\mathsf{Check}}$ lies in $V$, and $t_{\mathsf{Check}} < n_{\mathsf{Check}}/2$ guarantees one exists, so $r = \sum_{l \in V} r_l$ is uniform and independent of $c$; as multiplication by a fixed $c \neq 0$ is a bijection on $\mathbb{F}_p$, the released $r \cdot c$ is uniform on all of $\mathbb{F}_p$, and identically 0 when $c = 0$. This lets $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$ program the decryption from the commit bit alone, sampling uniformly on $\mathbb{F}_p$ when the era deviates, whereas releasing $c$ would disclose a linear function of $\alpha$. That sample is 0 with probability $1/p$, exactly the event that a deviating era is wrongly committed. The decision therefore agrees with the ideal $\mathsf{EraAbort}$ behaviour except with probability $2/p$ per era *attempt*; since each attempt consumes $m$ fresh computation committees there are at most $N_{\mathsf{Comp}}/m$ attempts, for a total of $2N_{\mathsf{Comp}}/(mp)$, which with the $\mathsf{Hyb}_1$ and $\mathsf{Hyb}_2$ terms and the soundness error remains negligible.

$\mathsf{Hyb}_4$. Identical to $\mathsf{Hyb}_3$, except that the simulator extracts the adversary's inputs using $\mathsf{sk}$ and forwards them to $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$. Intermediate era outputs are never needed, and indeed never available: $\mathcal{F}_{\mathsf{Era\text{-}MPC}}$ keeps each $\mathbf{y}_{i,j}$ internally and releases only progress notifications. The simulator instead carries its own dummy state across era boundaries, committing or rolling it back as the commit bit of $\mathcal{R}_{i,4}^{\mathsf{Check}}$ dictates; by $\mathsf{Hyb}_3$ it is distributed as the real state in $\mathcal{A}$'s view, and by MAC soundness it is committed only in eras carrying no adversarial offset.

Only in the final Output Phase are ideal values used. On receiving $(sid, \mathsf{Output}, \mathbf{y}_{l,m})$ the simulator must make $\mathcal{C}_{\mathsf{out}}$ reconstruct $\mathbf{y}_{l,m}$ although $\{\mathsf{ct}_{y_k}\}$ was broadcast while the dummy state was in force. It cannot revise those plaintexts, and knowing $\alpha$ only yields matching tags; it programs the decryption instead, which it may do because it emulates $\mathcal{F}_{\mathsf{TE}}$ and $\mathcal{R}_{l,4}^{\mathsf{Check}}$ never reveals the plaintexts it forwards. This is unnoticeable for two reasons. It is the only decryption whose plaintext depends on the output wires, every earlier release being either $y' = \sum_k r'_k \tilde{y}_k + R$ with $R$ an unopened mask, or $r \cdot c$, whose distribution depends on nothing but the commit bit. And the corrupt shares of the output wires, fixed in $\mathsf{Hyb}_2$, are consistent with $\mathbf{y}_{l,m}$ for exactly one assignment of the honest shares, which is the one used. The programmed decryption is therefore jointly distributed with the preceding ciphertexts, masked decryptions and corrupt shares as in $\mathsf{Hyb}_3$. This hybrid is exactly the ideal-world execution with $\mathcal{S}_{\mathsf{Era\text{-}MPC}}$, which completes the proof. □

## C Concrete Complexity Analysis

We report the computation and communication cost of $\Pi_{\mathsf{Era\text{-}MPC}}$ at four explicit layers: the one-time setup and input phase, per-sub-circuit evaluation, intra-era state handoffs, and era-boundary state transfer. Let $M_{i,j}$ denote the number of multiplication gates in $\mathbf{C}_{i,j}$, $\mathsf{int}_{i,j}$ the number of output wires of $\mathbf{C}_{i,j}$, and $K_i := \max(\mathsf{int}_{i-1,m}, \mathsf{int}_{i,m})$ the state-size parameter at era boundary $i$.

**Coin Toss protocol.** $n_1$ ciphertext-proof pairs broadcast; $n_1$ NIZK proofs generated and $n_1$ verified by the second committee. Costs no threshold decryptions.

**Setup and Input Phase** The setup phase invokes $t_{\mathsf{Prep}} + 1$ $\Pi_{\mathsf{CoinToss}}$ instances in $\mathcal{C}_{\mathsf{init2}}^{\mathsf{Check}}$ to generate $\mathsf{ct}_\alpha$ and the MAC-key polynomial coefficients. This is followed by two threshold decryptions: $\mathcal{C}_{\mathsf{init3}}^{\mathsf{Check}}$ decrypts $n_{\mathsf{Prep}}$ ciphertexts to deliver MAC key shares to $\mathcal{C}_0^{\mathsf{Prep}}$, and $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$ decrypts $n_{\mathsf{in}}$ ciphertexts to deliver input masks to $\mathcal{C}_0$ after running $2n_{\mathsf{in}}(n_{\mathsf{Comp}} - 1)$ further $\Pi_{\mathsf{CoinToss}}$ calls for the input mask shares.

The input sharing phase concludes with $t_{\mathsf{Prep}}$ $\Pi_{\mathsf{CoinToss}}$ calls in $\mathcal{R}_{0,1}^{\mathsf{Check}}$ for Era-1 key material and a third threshold decryption from $\mathcal{R}_{0,2}^{\mathsf{Check}}$, which simultaneously decrypts $2n_{\mathsf{in}}n_{\mathsf{Comp}} + n_{\mathsf{Prep}}$ ciphertexts: the initial SPDZ state to $\mathcal{R}_{1,1}^{\mathsf{Comp}}$ and MAC key shares to $\mathcal{R}_{1}^{\mathsf{Prep}}$. Communication totals $n_{\mathsf{in}}$ field-element broadcasts from $\mathcal{C}_0$ and $2n_{\mathsf{in}} \cdot n_{\mathsf{Comp}}$ ciphertexts forwarded to $\mathcal{R}_{0,1}^{\mathsf{Check}}$.

**Sub-circuit Evaluation and CheckOpen** For every sub-circuit $\mathbf{C}_{i,j}$, addition gates are computed locally with no communication. Each multiplication gate consumes one Beaver triple and requires two per-committee public reconstructions: every party broadcasts its share of $\epsilon = x - a$ and $\rho = y - b$, giving $2M_{i,j} \cdot n_{\mathsf{Comp}}$ total field-element broadcasts per sub-circuit. At the end of each sub-circuit, the **CheckOpen** step uses a commit-and-reveal coin-tossing protocol ($2n_{\mathsf{Comp}}$ messages) to produce a public random challenge; each of the $n_{\mathsf{Comp}}$ parties then encrypts a random linear combination of its MAC tag shares and attaches a NIZK proof of correct encryption, broadcasting $n_{\mathsf{Comp}}$ ciphertext-proof pairs to $\mathcal{R}_{i,1}^{\mathsf{Check}}$. There are no threshold decryptions and no $\Pi_{\mathsf{CoinToss}}$ invocations at this layer.

**Intra-era State Handoff** Each of the $n_{\mathsf{Comp}}$ parties in $\mathcal{R}_{i,j}^{\mathsf{Comp}}$ broadcasts its value share masked by $\beta_k^{i,j}$ and its MAC share masked by $\eta_k^{i,j}$, for each output wire, giving $2n_{\mathsf{Comp}} \cdot \mathsf{int}_{i,j}$ field-element broadcasts in total. Committee $\mathcal{R}_{i,j+1}^{\mathsf{Comp}}$ recovers its shares via $2\mathsf{int}_{i,j}$ local subtractions per party. There are no ciphertext operations, no threshold decryptions, and no $\Pi_{\mathsf{CoinToss}}$ calls at this layer.

**Era-boundary State Transfer** This is the most expensive layer, involving all four checkpoint committees.

- $\mathcal{R}_{i,m}^{\mathsf{Comp}}$ **(Commit-and-Encrypt).** Each of the $n_{\mathsf{Comp}}$ parties encrypts all $2\mathsf{int}_{i,m}$ share-plus-tag pairs and attaches NIZK proofs of correct encryption, broadcasting $2n_{\mathsf{Comp}} \cdot \mathsf{int}_{i,m}$ ciphertexts and proofs to $\mathcal{R}_{i,1}^{\mathsf{Check}}$.
- $\mathcal{R}_{i,1}^{\mathsf{Check}}$ **(MAC and Share Aggregation).** Verifies $2n_{\mathsf{Comp}}\mathsf{int}_{i,m} + mn_{\mathsf{Comp}}$ NIZK proofs. Homomorphically aggregates encrypted shares at cost $2\mathsf{int}_{i,m}(n_{\mathsf{Comp}} - 1)$ additions. Forms the blinded linear combination $\mathsf{ct}_{y+R}$ and its MAC counterpart with $2\mathsf{int}_{i,m}$ scalar multiplications and $2\mathsf{int}_{i,m}$ additions, and computes $\mathsf{ct}_\sigma$ with $m(n_{\mathsf{Comp}}+1)+m$ further homomorphic operations. Threshold-decrypts $\mathsf{ct}_{y+R}$: **1 decryption of 1 ciphertext**.
- $\mathcal{R}_{i,2}^{\mathsf{Check}}$ **(MAC Validation and Blinding).** Closes $\mathsf{ct}_c := (y' \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{ct}_\sigma$ with 1 scalar multiplication and 1 addition. Each of the $n_{\mathsf{Check}}$ parties forms $\mathsf{ct}_{c,l} := r_l \boxdot \mathsf{ct}_c$ with 1 scalar multiplication and generates 1 argument for $\mathsf{R}_{\mathsf{Scal}}$. No threshold decryption.
- $\mathcal{R}_{i,3}^{\mathsf{Check}}$ **(Aggregation).** Verifies $n_{\mathsf{Check}}$ arguments for $\mathsf{R}_{\mathsf{Scal}}$ and forms $\mathsf{ct}_{r\cdot c}$ with $n_{\mathsf{Check}} - 1$ homomorphic additions. Invokes $t_{\mathsf{Prep}}$ $\Pi_{\mathsf{CoinToss}}$ calls for fresh preprocessing key material and $2K_i(n_{\mathsf{Comp}} - 1)$ $\Pi_{\mathsf{CoinToss}}$ calls for per-wire masking ciphertexts. Threshold-decrypts $\mathsf{ct}_{r\cdot c}$: **1 decryption of 1 ciphertext**.
- $\mathcal{R}_{i,4}^{\mathsf{Check}}$ **(State Update or Rollback).** Decides commit or rollback on $r \cdot c \in \mathbb{F}_p$. Expands the compact state into per-party SPDZ shares at cost $2K_i(n_{\mathsf{Comp}} - 1)$ homomorphic additions. Evaluates the MAC-key polynomial at $n_{\mathsf{Prep}}$ points with $O(t_{\mathsf{Prep}} \cdot n_{\mathsf{Prep}})$ scalar multiplications. Threshold-decrypts the full next-era state and preprocessing key shares jointly to $\mathcal{R}_{i+1,1}^{\mathsf{Comp}}$ and $\mathcal{R}_{i+1}^{\mathsf{Prep}}$: **1 decryption of** $2K_i n_{\mathsf{Comp}} + n_{\mathsf{Prep}}$ **ciphertexts**.

**Summary per era boundary.** 3 threshold decryptions across 4 committees (of 1, 1, and $2K_i n_{\mathsf{Comp}} + n_{\mathsf{Prep}}$ ciphertexts); $n_{\mathsf{Check}}$ arguments for $\mathsf{R}_{\mathsf{Scal}}$ generated and verified; $t_{\mathsf{Prep}} + 2K_i(n_{\mathsf{Comp}} - 1)$ $\Pi_{\mathsf{CoinToss}}$ calls; $2\mathsf{int}_{i,m}(n_{\mathsf{Comp}} - 1) + 2K_i(n_{\mathsf{Comp}} - 1)$ homomorphic additions; $2\mathsf{int}_{i,m}$ scalar multiplications; $(2n_{\mathsf{Comp}}\mathsf{int}_{i,m} + mn_{\mathsf{Comp}})$ NIZK verifications. The dominant cost is the third decryption at $\mathcal{R}_{i,4}^{\mathsf{Check}}$ ($2K_i n_{\mathsf{Comp}} + n_{\mathsf{Prep}}$ ciphertexts) and $\mathsf{ct}_{\mathsf{St}_{\mathsf{Prv}}}$ ($2K_i \cdot n_{\mathsf{Comp}}$ ciphertexts).

## D Realizing the Preprocessing Functionality

This appendix gives Protocol $\Pi_{\mathsf{Prep}}$ of Section 3.4 in full, together with the circuit $\mathbf{C}_{\mathsf{Comp}}$ it evaluates and the protocols suitable for realizing $\mathcal{F}_{\mathsf{AMPC}}$. We present a conceptually simple, unoptimized version to highlight the underlying structure; further efficient instantiations are discussed in Section 4.

The circuit is specified over a finite field. Each preprocessing party contributes shares of the circuit inputs; upon evaluation via $\mathcal{F}_{\mathsf{AMPC}}$, parties obtain robust secret sharings of the outputs, which are forwarded to the appropriate committees as described in Figure 19. The circuit is defined as follows:

– $\mathbf{C}_{\mathsf{Comp}}$ : The circuit $\mathbf{C}_{\mathsf{Comp}}$ is defined as follows for one multiplication triple; a pair of *independent* masks $\beta$ and $\eta$, each shared once for the sub-circuit committee currently holding the wire and once for the next sub-circuit committee in the era, used for the threshold-decryption-free intra-era handoff between consecutive computation committees (these are the values $\beta_k^{i,j}, \eta_k^{i,j}$ of $\mathcal{F}_{\mathsf{Prep}}$, Figure 3), where $\beta$ masks the wire value and $\eta$ masks its tag; and an authenticated sharing $\langle 1 \rangle$ of the constant 1, generated once per computation committee and used to authenticate the public term of a multiplication. The era-boundary mask (the value $R$ of $\mathcal{F}_{\mathsf{Prep}}$) is not produced by this circuit; it is generated directly in encrypted form by the preprocessing committee, as described below:

- **Input:** For each $i \in [n_{\mathsf{Prep}}]$, the circuit takes as input

$$(\alpha_i,\ a_i,\ b_i,\ \beta_i,\ \eta_i,\ \{(\mathbf{R}_{x,i}, \mathbf{R}'_{x,i}, \rho_{x,i})\}_{x\in\{a,b,c,1\}},\ \{(\mathbf{R}^{(t)}_{\beta,i}, \mathbf{R}^{(t)}_{\eta,i}, \rho^{(t)}_{\beta,i}, \rho^{(t)}_{\eta,i})\}_{t\in\{1,2\}}).$$

- **Compute:**
  1. $\alpha := \mathsf{Rec}(\alpha_1, \ldots, \alpha_{n_{\mathsf{Prep}}})$
  2. $(a, b, \beta, \eta) := \sum_{i=1}^{n_{\mathsf{Prep}}} (a_i, b_i, \beta_i, \eta_i)$, $c := a \cdot b$
  3. $\mathbf{R}_x \leftarrow \sum_{i=1}^{n_{\mathsf{Prep}}} \mathbf{R}_{x,i}$, $\mathbf{R}'_x \leftarrow \sum_{i=1}^{n_{\mathsf{Prep}}} \mathbf{R}'_{x,i}$ for $x \in \{a, b, c, 1\}$
  4. $\mathbf{R}^{(t)}_\beta \leftarrow \sum_{i=1}^{n_{\mathsf{Prep}}} \mathbf{R}^{(t)}_{\beta,i}$, $\mathbf{R}^{(t)}_\eta \leftarrow \sum_{i=1}^{n_{\mathsf{Prep}}} \mathbf{R}^{(t)}_{\eta,i}$ for $t \in \{1, 2\}$
  5. $\rho_x \leftarrow \sum_{i=1}^{n_{\mathsf{Prep}}} \rho_{x,i}$ for $x \in \{a, b, c, 1\}$; $\rho^{(t)}_\beta \leftarrow \sum_{i=1}^{n_{\mathsf{Prep}}} \rho^{(t)}_{\beta,i}$ and $\rho^{(t)}_\eta \leftarrow \sum_{i=1}^{n_{\mathsf{Prep}}} \rho^{(t)}_{\eta,i}$ for $t \in \{1, 2\}$
  6. $\langle x \rangle \leftarrow (\mathsf{AddShare}(x; \mathbf{R}_x),\ \mathsf{AddShare}(\alpha \cdot x; \mathbf{R}'_x))$ for $x \in \{a, b, c, 1\}$
  7. $[\beta]^{(t)} \leftarrow \mathsf{AddShare}(\beta; \mathbf{R}^{(t)}_\beta)$, $[\eta]^{(t)} \leftarrow \mathsf{AddShare}(\eta; \mathbf{R}^{(t)}_\eta)$ for $t \in \{1, 2\}$ // $\eta$ is sampled, not $\alpha\beta$
  8. $[\![\langle x \rangle]\!] \leftarrow \mathsf{Sh}(\langle x \rangle; \rho_x)$ for $x \in \{a, b, c, 1\}$
  9. $[\![[\beta]]\!]^{(t)} \leftarrow \mathsf{Sh}([\beta]^{(t)}; \rho^{(t)}_\beta)$, $[\![[\eta]]\!]^{(t)} \leftarrow \mathsf{Sh}([\eta]^{(t)}; \rho^{(t)}_\eta)$ for $t \in \{1, 2\}$
- **Output:**

$$([\![\langle a \rangle]\!],\ [\![\langle b \rangle]\!],\ [\![\langle c \rangle]\!],\ [\![\langle 1 \rangle]\!],\ [\![[\beta]]\!]^{(1)},\ [\![[\eta]]\!]^{(1)},\ [\![[\beta]]\!]^{(2)},\ [\![[\eta]]\!]^{(2)}).$$

### D.1 Preprocessing Protocol

We now describe $\Pi_{\mathsf{Prep}}$ in full. The protocol is written in the $\mathcal{F}_{\mathsf{AMPC}}$-hybrid model to simplify exposition; $\mathcal{F}_{\mathsf{AMPC}}$ serves as a local abstraction for evaluating the fixed circuits above and is not the ideal functionality realized by the overall MPC construction.

The protocol proceeds in three stages: initialization of the authenticated MAC key, generation of the authenticated input masks by direct contribution, and computation of Beaver triples and masking values via $\mathbf{C}_{\mathsf{Comp}}$.

Protocol $\Pi_{\mathsf{Prep}}$ realises $\mathcal{F}_{\mathsf{Prep}}$ as stated in Theorem 2; the complete proof can be found in Section E.

**Cost of the Authenticated Randomness Arguments:** The arguments accompanying the input masks and the era-boundary mask prove a conjunction: that $\mathsf{ct}_{r_i}$ encrypts a known $r_i$, and that $\mathsf{ct}_{\gamma(r_i)}$ is the same $r_i$ applied to $\mathsf{ct}_\alpha$. Instantiated with a general-purpose non-interactive argument system this would be expensive, but it is never necessary to do so. For every concrete linearly homomorphic scheme we are aware of, the relation is a statement about a linear map on ciphertexts and admits a short $\Sigma$-protocol with a shared witness, made non-interactive by the Fiat–Shamir transform. For group-based schemes such as ElGamal the two clauses are equality-of-discrete-logarithm statements and compose directly; for Paillier [54] and class groups [16,23] the corresponding arguments are those already used in CDN-style protocols [17,26], subject to the usual care over challenge spaces in groups of unknown order. The cost is therefore a constant number of group operations per contribution, and is independent of the circuit being evaluated.

**Public Parameters:** Committee pools $\mathcal{P}_{\mathsf{Comp}} := \{\mathcal{C}_1^{\mathsf{Comp}}, \cdots, \mathcal{C}_{N_{\mathsf{Comp}}}^{\mathsf{Comp}}\}, \mathcal{P}_{\mathsf{Prep}} := \{\mathcal{C}_1^{\mathsf{Prep}}, \cdots, \mathcal{C}_{N_{\mathsf{Prep}}}^{\mathsf{Prep}}\}, \mathcal{P}_{\mathsf{Check}} := \{\mathcal{C}_1^{\mathsf{Check}}, \cdots, \mathcal{C}_{N_{\mathsf{Check}}}^{\mathsf{Check}}\}$ as described in Section 3; circuit $\mathbf{C}$ partitioned into sub-circuits $\{\mathbf{C}_{i,j}\}_{i\in[l],j\in[m]}$; input committee $\mathcal{C}_{\mathsf{in}} := \mathcal{C}_0$.
**Building Blocks:** Ideal functionality $\mathcal{F}_{\mathsf{AMPC}}$ (Figure 9) used to compute any arithmetic circuit $\mathbf{C}$, a CPA-secure encryption scheme $\mathsf{Lin\text{-}PKE}$ as in Definition 13, a $(t, n)$-threshold secret sharing scheme $(\mathsf{Sh}, \mathsf{Rec})$, a ZK Argument $\Pi_{\mathsf{NIZK}}$ as in Definition 14.
**PrepInput:** Each $P_i \in \mathcal{C}_0^{\mathsf{Prep}}$ with input $(\mathsf{pk}, \mathsf{ct}_\alpha, \alpha_i)$, does the following for each $P_j \in \mathcal{C}_0$:

1. Sample $r_{j,i} \leftarrow\!\!\$\ \mathbb{F}_p$, compute $\mathsf{ct}_{r_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}, r_{j,i}; \rho_{j,i})$ and $\mathsf{ct}_{\gamma(r_{j,i})} := (r_{j,i} \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; s_{j,i})$.
2. Compute $\pi_{r_{j,i}}^{\mathsf{Auth}} \leftarrow \mathsf{Prove}((\mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_{r_{j,i}}, \mathsf{ct}_{\gamma(r_{j,i})}), (r_{j,i}, \rho_{j,i}, s_{j,i}))$, and broadcast $(\mathsf{ct}_{r_{j,i}}, \mathsf{ct}_{\gamma(r_{j,i})}, \pi_{r_{j,i}}^{\mathsf{Auth}})$ to $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$.
3. Parties in $\mathcal{C}_{\mathsf{init4}}^{\mathsf{Check}}$ verify the arguments, let $V_j$ be the indices that verify, and output $\mathsf{ct}_{r_j} := \boxplus_{i\in V_j} \mathsf{ct}_{r_{j,i}}$ and $\mathsf{ct}_{\gamma(r_j)} := \boxplus_{i\in V_j} \mathsf{ct}_{\gamma(r_{j,i})}$.

**PrepCompute:** Each $P_k \in \mathcal{C}_{\mathsf{PrepCtr}}^{\mathsf{Prep}}$ with input $(i, \mathsf{CompCtr}, \mathsf{CheckCtr}, \mathsf{pk}, \mathsf{ct}_\alpha, \alpha_k)$ does the following for each $j \in [m]$, with respect to each multiplication gate of sub-circuit $\mathbf{C}_{i,j}$ for triples and intermediate-output masks, and once per era for the era-boundary mask $R$:

1. Sample $\mathsf{In}_k$ as below and send $(sid, \mathsf{input}, P_k, \mathsf{varid}, (\alpha_k, \mathsf{In}_k))$ to $\mathcal{F}_{\mathsf{AMPC}}$:
$$\mathsf{In}_k := (a_k, b_k, \beta_k, \eta_k, \{\mathbf{R}_{x,k}, \mathbf{R}'_{x,k}, \rho_{x,k}\}_{x\in\{a,b,c,1\}}, \{\mathbf{R}_{\beta,k}^{(t)}, \mathbf{R}_{\eta,k}^{(t)}, \rho_{\beta,k}^{(t)}, \rho_{\eta,k}^{(t)}\}_{t\in\{1,2\}}).$$
2. Evaluate $\mathbf{C}_{\mathsf{Comp}}$ using $\mathcal{F}_{\mathsf{AMPC}}$ using appropriate calls to $\mathsf{add}$ and $\mathsf{multiply}$.
3. Receive shares $(\llbracket\langle a\rangle\rrbracket_k, \llbracket\langle b\rangle\rrbracket_k, \llbracket\langle c\rangle\rrbracket_k, \llbracket\langle 1\rangle\rrbracket_k, \llbracket[\beta]\rrbracket_k^{(1)}, \llbracket[\eta]\rrbracket_k^{(1)}, \llbracket[\beta]\rrbracket_k^{(2)}, \llbracket[\eta]\rrbracket_k^{(2)})$ from $\mathcal{F}_{\mathsf{AMPC}}$.
4. Sample $R_k \leftarrow\!\!\$\ \mathbb{F}_p$, compute $\mathsf{ct}_{R_k} \leftarrow \mathsf{Enc}(\mathsf{pk}, R_k; \rho_{R,k})$ and $\mathsf{ct}_{\gamma(R_k)} := (R_k \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; s_{R,k})$, together with the argument $\pi_{R_k}^{\mathsf{Auth}} \leftarrow \mathsf{Prove}((\mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_{R_k}, \mathsf{ct}_{\gamma(R_k)}), (R_k, \rho_{R,k}, s_{R,k}))$, and broadcast $(\mathsf{ct}_{R_k}, \mathsf{ct}_{\gamma(R_k)}, \pi_{R_k}^{\mathsf{Auth}})$ to $\mathcal{C}_{\mathsf{CheckCtr}}^{\mathsf{Check}}$.
5. Send shares $\llbracket\langle a\rangle_{k'}\rrbracket_k, \llbracket\langle b\rangle_{k'}\rrbracket_k, \llbracket\langle c\rangle_{k'}\rrbracket_k, \llbracket\langle 1\rangle_{k'}\rrbracket_k, \llbracket[\beta]_{k'}\rrbracket_k^{(1)}, \llbracket[\eta]_{k'}\rrbracket_k^{(1)}$ to $P_{k'} \in \mathcal{C}_{\mathsf{CompCtr}+j-1}^{\mathsf{Comp}}$; if $j < m$, additionally send $\llbracket[\beta]_{k'}\rrbracket_k^{(2)}, \llbracket[\eta]_{k'}\rrbracket_k^{(2)}$ to $P_{k'} \in \mathcal{C}_{\mathsf{CompCtr}+j}^{\mathsf{Comp}}$.
6. Parties in $\mathcal{C}_{\mathsf{CompCtr}+j-1}^{\mathsf{Comp}}$, and parties in $\mathcal{C}_{\mathsf{CompCtr}+j}^{\mathsf{Comp}}$ if $j < m$, reconstruct and output their SPDZ-shares.
7. Parties in $\mathcal{C}_{\mathsf{CheckCtr}}^{\mathsf{Check}}$ verify the arguments, let $V$ be the indices that verify, and output $\mathsf{ct}_R := \boxplus_{k\in V} \mathsf{ct}_{R_k}$ and $\mathsf{ct}_{\gamma(R)} := \boxplus_{k\in V} \mathsf{ct}_{\gamma(R_k)}$.

**Figure 19:** Protocol $\Pi_{\mathsf{Prep}}$ in the $\mathcal{F}_{\mathsf{AMPC}}$-hybrid

### D.2 Realizing $\mathcal{F}_{\mathsf{AMPC}}$ for Efficient Instantiations of $\Pi_{\mathsf{Prep}}$

The efficiency of $\Pi_{\mathsf{Prep}}$ depends primarily on the choice of protocol realizing $\mathcal{F}_{\mathsf{AMPC}}$, since it accounts for the bulk of the communication and computation. The goal is to select a protocol that can compute two large batches of multiplications needed to generate the authenticated Beaver triples and random values as efficiently as possible. The two batches account for the triple arithmetic only; the invocations of $\mathsf{Sh}$ in $\mathbf{C}_{\mathsf{Comp}}$ and of $\mathsf{Rec}$ on the key shares carry a further cost that depends on the instantiation chosen in Remark 12, and which the next paragraph shows how to avoid for $\mathsf{Sh}$. Since preprocessing committees operate with honest majority, information-theoretic MPC protocols are a natural choice.

**Computation Domain Compatibility:** Since $\Pi_{\mathsf{Prep}}$ outputs robust secret shares of the correlated data to computation committees, it can be more efficiently instantiated by an MPC protocol that performs operations over inputs shared by a robust secret sharing scheme and produces outputs shared by the same scheme. If $\mathbf{C}_{\mathsf{Comp}}$ is computed by an MPC protocol that natively produces circuit outputs that are robustly secret shared, all invocations of $\mathsf{Sh}()$ in Steps 8 and 9 of $\mathbf{C}_{\mathsf{Comp}}$ can be eliminated. Foregoing these operations reduces not only the amount of computation and communication among the preprocessing committee evaluating these circuits but also the amount of randomness they must use as input. A simple way to achieve this is working with $n_{\mathsf{Prep}} \geq 3t_{\mathsf{Prep}} + 1$ and using MPC protocols that evaluate circuits over values secret shared via Shamir secret sharing, which is known to be robust in the $n_{\mathsf{Prep}} \geq 3t_{\mathsf{Prep}} + 1$ regime. Alternatively, MPC protocols using alternative secret sharing schemes that can be easily converted to Shamir are also a viable option.

**Communication-efficient information theoretical MPC protocols:** If the entire preprocessing can be done by a single committee, communication-efficient MPC protocols such as [38, 39, 46] can be employed to produce the necessary correlated randomness with a low amortized cost. In case a

preprocessing phase with dynamic committees with maximal fluidity is required, protocols such as [12, 40] can be employed, although at a much higher cost, which necessarily includes resharing the entire state among committees. A middle ground can be achieved with dynamic committee MPC protocols where each committee executes more than one round such as [33], avoiding constant resharing. In particular, the protocols of [33] can evaluate the required multiplications in $\mathbf{C}_{\mathsf{Comp}}$ employing only 2 committees and thus requiring only one resharing.

## E Proof for Theorem 2

We show that the protocol presented in Figure 19 realizes $\mathcal{F}_{\mathsf{Prep}}$ in Figure 3 with guaranteed output delivery in the presence of a malicious, static, rushing adversary $\mathcal{A}$ which can corrupt parties as follows: For any committee $\mathcal{C}_x$, let $\mathcal{I}_x \subset \mathcal{C}_x$ denote the set of corrupted parties. For any era $i \in [l]$:

- Preprocessing: $|\mathcal{I}_i^{\mathsf{Prep}}| < \frac{|\mathcal{R}_i^{\mathsf{Prep}}|}{2}$.
- Checkpoint: $|\mathcal{I}_{i,k}^{\mathsf{Check}}| < \frac{|\mathcal{R}_{i,k}^{\mathsf{Check}}|}{2}$ for each $k \in [4]$.
- Computation: $|\mathcal{I}_{i,j}^{\mathsf{Comp}}| < |\mathcal{R}_{i,j}^{\mathsf{Comp}}|$ for each $j \in [m]$

The proof for this lemma follows from a PPT simulation algorithm $\mathcal{S}_{\mathsf{Prep}}$ that we define for the protocol $\Pi_{\mathsf{Prep}}$, where $\mathcal{S}_{\mathsf{Prep}}$ also emulates the ideal functionality $\mathcal{F}_{\mathsf{AMPC}}$ (Figure 9) along with the messages that the adversary needs to receive from it. The simulator $\mathcal{S}_{\mathsf{Prep}}$, detailed in Figures 20,21, also uses the SHVZK simulator $\mathsf{Sim}_{\mathsf{NIZK}}$ for the $\Pi_{\mathsf{NIZK}}$ system in a black-box way.

**PrepInput:** Do the following for each $P_j \in \mathcal{C}_0$:

1. Receive $\mathsf{pk}$ and the ciphertexts $(\mathsf{ct}_{r_j}, \mathsf{ct}_{\gamma(r_j)})$ from $\mathcal{F}_{\mathsf{Prep}}$. Send $(\mathsf{PrepInput}, sid, \mathcal{C}_0, (\mathsf{pk}, \alpha_i))$ to $\mathcal{F}_{\mathsf{Prep}}$ on behalf of every $P_i \in \mathcal{I}_0^{\mathsf{Prep}}$.
2. Fix an honest index $i^* \in \mathcal{C}_0^{\mathsf{Prep}} \setminus \mathcal{I}_0^{\mathsf{Prep}}$. For every other honest $P_i$, sample $r_{j,i} \leftarrow\!\!\$\ \mathbb{F}_p$ and follow the protocol: $\mathsf{ct}_{r_{j,i}} \leftarrow \mathsf{Enc}(\mathsf{pk}, r_{j,i}; \rho_{j,i})$ and $\mathsf{ct}_{\gamma(r_{j,i})} := (r_{j,i} \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; s_{j,i})$.
3. Receive $(\mathsf{ct}_{r_{j,i}}, \mathsf{ct}_{\gamma(r_{j,i})}, \pi^{\mathsf{Auth}}_{r_{j,i}})$ from $\mathcal{A}$ for every $P_i \in \mathcal{I}_0^{\mathsf{Prep}}$, verify the arguments, and let $V_j$ be the indices that verify together with the honest indices.
4. Set $\mathsf{ct}_{r_{j,i^*}} := \mathsf{ct}_{r_j} \boxplus (-1 \boxdot \boxplus_{i \in V_j \setminus \{i^*\}} \mathsf{ct}_{r_{j,i}})$ and likewise $\mathsf{ct}_{\gamma(r_{j,i^*})}$ against $\mathsf{ct}_{\gamma(r_j)}$, so that the aggregate over $V_j$ matches the ciphertexts received from $\mathcal{F}_{\mathsf{Prep}}$.
5. Generate simulated arguments $\tilde{\pi}^{\mathsf{Auth}}_{r_{j,i}} \leftarrow \mathsf{Sim}_{\mathsf{NIZK}}(\mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_{r_{j,i}}, \mathsf{ct}_{\gamma(r_{j,i})})$ for all honest $P_i$, and broadcast the honest contributions and arguments to $\mathcal{A}$.

**Figure 20:** Simulator $\mathcal{S}_{\mathsf{Prep}}$ for the protocol $\Pi_{\mathsf{Prep}}$ - **PrepInput**

*Binding of the authenticated contributions.* Computational soundness of $\Pi_{\mathsf{NIZK}}$ for $\mathsf{R}_{\mathsf{Auth}}$ suffices here; no knowledge extractor is required. For every accepted contribution $(\mathsf{ct}_{r_{j,i}}, \mathsf{ct}_{\gamma(r_{j,i})}, \pi^{\mathsf{Auth}}_{r_{j,i}})$ soundness gives that a witness $(\mu_i, \rho_i, s_i)$ *exists* with $\mathsf{ct}_{r_{j,i}} = \mathsf{Enc}(\mathsf{pk}, \mu_i; \rho_i)$ and $\mathsf{ct}_{\gamma(r_{j,i})} = (\mu_i \boxdot \mathsf{ct}_\alpha) \boxplus \mathsf{Enc}(\mathsf{pk}, 0; s_i)$, and existence alone fixes both plaintexts: $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{r_{j,i}}) = \mu_i$ and $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\gamma(r_{j,i})}) = \alpha\mu_i$. Since $\mathsf{Dec}$ is linear over $\boxplus$, the aggregate over the verifying set satisfies $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{\gamma(r_j)}) = \alpha \cdot \mathsf{Dec}(\mathsf{sk}, \mathsf{ct}_{r_j})$, for adaptively chosen $\mu_i$ and although no party holds $\alpha$ in the clear. The same applies to the contributions forming $\mathsf{ct}_R, \mathsf{ct}_{\gamma(R)}$. The outputs of $\mathcal{F}_{\mathsf{Prep}}$ delivered in encrypted form are therefore correctly authenticated by construction.

*Guaranteed output delivery.* A corrupt party that withholds its contribution or supplies a failing argument is excluded from $V_j$ (resp. $V$), and the remaining contributions are aggregated regardless. Since $t_{\mathsf{Prep}} < n_{\mathsf{Prep}}/2$, at least one honest contribution always survives, so the aggregate is uniformly distributed and the output is delivered whatever the corrupt parties do. The shares produced through $\mathcal{F}_{\mathsf{AMPC}}$ inherit guaranteed output delivery from $\mathcal{F}_{\mathsf{AMPC}}$ itself.

The indistinguishability of the simulator's output from the adversary's view in the real execution is established through the following sequence of hybrids. While the information-theoretic components of the protocol (the $\mathcal{F}_{\mathsf{AMPC}}$ shares) are perfectly simulated, the security of the broadcast ciphertexts and proofs rely on the underlying cryptographic primitives.

**PrepCompute:** For each multiplication gate of sub-circuit $\mathbf{C}_{i,j}$, $j \in [m]$, and once per era for $R$:

1. Receive $\mathsf{pk}$ from $\mathcal{F}_{\mathsf{Prep}}$.
2. For emulating $\mathcal{F}_{\mathsf{AMPC}}$:
   (a) Receive $(sid, \mathsf{input}, P_k, \mathsf{varid}, (\alpha_i, \mathsf{In}_k))$ from $\mathcal{A}$ for every $P_k \in \mathcal{I}_i^{\mathsf{Prep}}$ on behalf of $\mathcal{F}_{\mathsf{AMPC}}$. Receive $\mathsf{pk}$ from $\mathcal{F}_{\mathsf{Prep}}$. Send $(\mathsf{PrepCompute}, sid, i, \mathsf{CompCtr}, \mathsf{CheckCtr}, \alpha_k)$ to $\mathcal{F}_{\mathsf{Prep}}$ on behalf of every $P_k \in \mathcal{I}_i^{\mathsf{Prep}}$.
   (b) Receive $\mathsf{add}$ or $\mathsf{multiply}$ calls from $\mathcal{A}$ appropriately on behalf of $\mathcal{F}_{\mathsf{AMPC}}$.
   (c) For all corrupt parties $P_k \in \mathcal{I}_i^{\mathsf{Prep}}$, send random shares $(\llbracket\langle a\rangle\rrbracket_k, \llbracket\langle b\rangle\rrbracket_k, \llbracket\langle c\rangle\rrbracket_k, \llbracket\langle 1\rangle\rrbracket_k, \llbracket[\beta]\rrbracket_k^{(1)}, \llbracket[\eta]\rrbracket_k^{(1)}, \llbracket[\beta]\rrbracket_k^{(2)}, \llbracket[\eta]\rrbracket_k^{(2)})$ to $\mathcal{A}$.
3. Receive $\mathsf{ct}_R, \mathsf{ct}_{\gamma(R)}$, along with shares $\langle a\rangle_k, \langle b\rangle_k, \langle c\rangle_k, \langle 1\rangle_k, [\beta]_k^{(1)}, [\eta]_k^{(1)}$ for every $P_k \in \mathcal{I}_{i,j}^{\mathsf{Comp}}$, and shares $[\beta]_k^{(2)}, [\eta]_k^{(2)}$ for every $P_k \in \mathcal{I}_{i,j+1}^{\mathsf{Comp}}$ when $j < m$.
4. For every $P_k \in \mathcal{R}_{i,j}^{\mathsf{Comp}}$, compute $\{\llbracket\langle x\rangle_k\rrbracket\}_{x\in\{a,b,c,1\}}$, $\llbracket[\beta]\rrbracket_k^{(1)}$ and $\llbracket[\eta]\rrbracket_k^{(1)}$ for $\mathcal{R}_i^{\mathsf{Prep}}$ consistent with the shares already sent to $\mathcal{A}$ in Step 2(c), and send these to $\mathcal{A}$. If $j < m$, do the same for $\llbracket[\beta]\rrbracket_k^{(2)}, \llbracket[\eta]\rrbracket_k^{(2)}$ with respect to every $P_k \in \mathcal{R}_{i,j+1}^{\mathsf{Comp}}$.
5. Simulate the era-boundary mask exactly as in Figure 20: fix an honest index $k^*$, follow the protocol for every other honest $P_k \in \mathcal{R}_i^{\mathsf{Prep}} \setminus \mathcal{I}_i^{\mathsf{Prep}}$, receive and verify the corrupt contributions to obtain $V$, and set $P_{k^*}$'s contribution so that $\boxplus_{k\in V}\mathsf{ct}_{R_k} = \mathsf{ct}_R$ and $\boxplus_{k\in V}\mathsf{ct}_{\gamma(R_k)} = \mathsf{ct}_{\gamma(R)}$.
6. Generate simulated arguments $\tilde{\pi}_{R_k}^{\mathsf{Auth}} \leftarrow \mathsf{Sim}_{\mathsf{NIZK}}(\mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_{R_k}, \mathsf{ct}_{\gamma(R_k)})$ for all honest $P_k$, and broadcast the honest contributions and arguments to $\mathcal{A}$.

**Figure 21:** Simulator $\mathcal{S}_{\mathsf{Prep}}$ for the protocol $\Pi_{\mathsf{Prep}}$ - **PrepCompute**

*Hybrid Sequence.*

- $\mathsf{Hyb}_0$: The real-world execution of $\Pi_{\mathsf{Prep}}$ in the $\mathcal{F}_{\mathsf{AMPC}}$-hybrid model.
- $\mathsf{Hyb}_1$: Identical to $\mathsf{Hyb}_0$, except that the NIZK proofs for all honest parties are replaced by simulated proofs generated via $\mathsf{Sim}_{\mathsf{NIZK}}$. Indistinguishability follows from the statistical $\mathsf{SHVZK}$ property of the proof system.
- $\mathsf{Hyb}_2$: Identical to $\mathsf{Hyb}_1$, except that the contribution of the designated honest party $P_{i^*}$ is no longer generated by running the protocol on a freshly sampled summand, but is set to the residual against the ciphertexts received from $\mathcal{F}_{\mathsf{Prep}}$. The remaining honest parties still follow the protocol. This hybrid is the ideal execution with $\mathcal{S}_{\mathsf{Prep}}$.

*From* $\mathsf{Hyb}_1$ *to* $\mathsf{Hyb}_2$. The two hybrids differ only in how $P_{i^*}$'s contribution is produced, so it suffices to compare the joint distribution of the honest contributions in each. Write $H$ for the honest indices.

For the plaintexts: in $\mathsf{Hyb}_1$ the summands $\{r_{j,i}\}_{i\in H}$ are uniform and independent, so their sum is uniform. In $\mathsf{Hyb}_2$ the summands $\{r_{j,i}\}_{i\in H\setminus\{i^*\}}$ are uniform and independent and their sum with $P_{i^*}$'s is the mask $r_j$ that $\mathcal{F}_{\mathsf{Prep}}$ sampled uniformly, which fixes $P_{i^*}$'s implicit summand as $r_j - \sum_{i\neq i^*} r_{j,i}$; this is again uniform and independent of the others. The two families are therefore identically distributed, and the tags follow, being $\alpha$ times the corresponding summand in both.

For the ciphertexts: in $\mathsf{Hyb}_1$ every honest contribution is a fresh encryption, while in $\mathsf{Hyb}_2$ the one belonging to $P_{i^*}$ is obtained homomorphically. These coincide because the randomness translates through $\boxplus$. Writing $\rho_\star$ for the randomness $\mathcal{F}_{\mathsf{Prep}}$ used on $\mathsf{ct}_{r_j}$, the residual carries randomness $\rho_\star - \sum_{i\neq i^*}\rho_{j,i}$, and $\rho_\star$ is uniform and independent of everything in $\mathcal{A}$'s view, so the residual is distributed exactly as a fresh encryption of its plaintext. The same holds for the tag component, where the rerandomizing term $\mathsf{Enc}(\mathsf{pk}, 0; s_{j,i})$ of $\mathsf{R}_{\mathsf{Auth}}$ is what makes each honest tag ciphertext's randomness uniform in the first place; without it $\mathsf{ct}_{\gamma(r_{j,i})}$ would be a deterministic function of $r_{j,i}$ and $\mathsf{ct}_\alpha$, and no residual could be consistent with it.

The two hybrids are therefore statistically close. No appeal to IND-CPA is needed, and in particular there is no reduction that would have to manufacture an encryption of $\alpha\mu$ from an encryption of an unknown $\mu$.

Finally, the shares handed to $\mathcal{A}$ while emulating $\mathcal{F}_{\mathsf{AMPC}}$: corrupt preprocessing parties receive at most $t_{\mathsf{Prep}}$ shares of each robustly shared output, which by the $t$-security of $(\mathsf{Sh}, \mathsf{Rec})$ are distributed independently of the shared value, so sending random shares in Step 2(c) and reconciling them in Step 4 is perfect.