Accountable Asynchronous Multi-Party Computation

Pierre Civit Rachid Guerraoui
EPFL (Ecole Polytechnique Fédérale de Lausanne)
{pierre.civit, rachid.guerraoui}@epfl.ch

***Abstract*—In non-synchronous networks, classic partition arguments imply that any t-resilient protocol among n parties cannot ensure safety for many meaningful functionalities once the number of corruptions reaches $f \geq n - 2t$. This motivates building in *accountability* to detect (and deter) safety violations.**

We present the first accountable asynchronous MPC (AAMPC) protocol that securely evaluates any arithmetic circuit $\mathcal{C}$ (asynchronously computable by a trusted third party). Our protocol:

(1) Ensures all target hyperproperties (correctness, privacy, input-independence, and guaranteed output delivery) whenever $f \leq t < n/3$.

(2) Provides strong accountability for $f \in (t, t_{\text{acc}}]$ with $t_{\text{acc}} < n - t$: either (i) all hypersafety properties continue to hold (without guaranteed output delivery), or (ii) every honest party obtains publicly verifiable evidence implicating at least $n - 2t$ faulty processes.

The construction follows the standard offline/online paradigm and assumes only a transparent setup: a bulletin-board public key infrastructure (PKI) and a common random string (CRS).

Our main technical contribution is an accountable additively homomorphic high-threshold asynchronous complete (verifiable) secret sharing functionality with amortized linear communication for both sharing and reconstruction. This yields an efficient online phase with $O\big(\mathsf{Depth}(\mathcal{C})\big)$ latency and amortized $O(|\mathcal{C}|n)$ communication.

We additionally provide a constant-round offline phase with cubic communication per generated Beaver triple.

Our results are formalized and proven in the Accountable Universal Composability (AUC) framework (S&P 2023), an extension of UC designed to support modular analysis of accountability guarantees.

1. Introduction

MPC (In)Feasibility. In secure multi-party computation (MPC) [1], [2], a set Ψ of n processes $p_1, \ldots, p_n$ collectively emulate a functionality (e.g., distributed machine learning, voting, auctions, payment systems) while tolerating up to f faulty processes, ranging from crashes to fully malicious behavior. This emulation is defined relative to an ideal trusted third party that would (a) privately receive processes' inputs, (b) locally evaluate the corresponding arithmetic circuit $\mathcal{C}$, and (c) privately return each output to the designated recipient. The resulting protocol should satisfy the following (hyper)properties: (1) *Correctness* (outputs are consistent with inputs and $\mathcal{C}$); (2) *Privacy* (no coalition learns more than what follows from its inputs, outputs, and the circuit); (3) *Independence of inputs* (inputs are chosen independently, not implied by privacy); (4) *Guaranteed output delivery* (all correct processes eventually obtain their outputs); or, under relaxation, (5) *Fairness* (corrupted parties receive outputs only if honest parties do too).

The classical impossibility of fairness with fewer than $n/2$ honest parties [3] motivated the study [4], [5], [6], [7], [8], [9], [10], [11], [12], [13] of synchronous MPC with identifiable abort (ID-MPC). Originating in the covert security model [14], [15], this line of work culminated in the (synchronous) publicly accountable MPC protocol of [16], which achieves all hyperproperties if $f < n/2$, and otherwise provides publicly verifiable identification of at least $\lceil n/2 \rceil$ faulty parties.

In the asynchronous setting, any functionality (asynchronously) computable by a trusted party can be emulated with up to $n/3$ malicious faults [17], or up to $n/2$ if faults are non-equivocating [18].[1] Still, classic partitioning arguments [24] show that many functionalities become impossible to compute once message delivery is arbitrarily delayed and $f \geq n/3$. These limits affect not only liveness but also safety, which can be violated when $f \geq n - 2t$, where t is the resilience threshold of the protocol. Perhaps surprisingly, there is essentially no asynchronous analogue of ID-MPC protocols.

Easy Accountability. To address the inherent limitation of the asynchronous setting, researchers have explored the concept of *accountability* [25]. In accountable protocols, violations of critical properties (e.g., agreement in a blockchain) result in the detection and non-repudiable attribution of misbehavior to specific malicious processes.

A recent line of work has focused on the goal of making classic agreement primitives accountable: if safety is violated due to an over-corrupted system, correct processes should be able to identify and publicly expose the culprits. In particular, the $\mathcal{ABC}$-like approach [26], [27], [28] introduces the *accountable confirmer*: a simple, low-overhead, and plug-and-play functionality denoted $\mathcal{F}_{\text{conf}}^{\text{acc}}$. In a nutshell,

1. The flaw in [18], noted in [19], can be addressed by using an $n/2$-resilient agreement on a core set (ACS) [20], [21], [22], avoiding reduction to binary agreement with strong unanimity, which is impossible even with a trusted party [19]. The same applies to [23].

the accountable confirmer functionality instructs processes to ratify a (pre-)agreement by attaching non-repudiable signatures before deciding on a value. In the nominal case, i.e., when $f \leq t$, this adds only a single round of latency. In the degraded case ($f > t$), however, any disagreement is eventually traced back to a significant set of faulty processes that signed conflicting ratification messages for different values. The confirmer can be sequentially composed with any easily accountable agreement functionality $\mathcal{F}_{\mathrm{ea}}$, such as any flavor of Byzantine Agreement, including Multi-value Validated Byzantine Agreement $\mathcal{F}_{\mathrm{mvba}}$ [29] or (indexed) Agreement on a Core Set $\mathcal{F}_{\text{idx-acs}}$ [30], [31], Reliable Broadcast $\mathcal{F}_{\mathrm{rbc}}$ [32], or one-to-many zero-knowledge proof of knowledge $\mathcal{F}_{\mathrm{zkPoK}}^{1:\mathrm{M}}$ [33], to yield its accountable counterpart $\mathcal{F}_{\mathrm{ea}}^{\mathrm{acc}}$. These accountable functionalities extend the guarantees of each functionality by ensuring that, in case of a disagreement caused by excessive corruption, all honest processes eventually obtain externally verifiable evidence identifying a significant set of faulty parties. After-the-fact detection is meaningful in repeated or institutional deployments: it can support invalidating an execution, excluding faulty parties, or imposing contractual, regulatory, or reputational sanctions. In the concrete ABC-style realization, once a safety violation has occurred, the evidence pieces are already in flight and are disseminated within a constant number of additional rounds.

General Accountability. Beyond such task-specific transformations, researchers have also proposed generic compilers for accountability. Early systems [34], [35], [36] like PeerReview relied on failure detector-inspired [37] mechanisms, resulting in permanent suspicion without irrefutable evidence. More recent approaches, such as the $\tau_{zk\text{-}scr}$ compiler [38], revisit classical compilers akin to Bracha [32] and Goldreich, Micali, and Wigderson [39], later formalized in the CLOS compiler by Canetti, Lindell, Ostrovsky, and Sahai [33]. At its core, this approach, informally referred to as "BCLOS", relies on the commit-and-prove paradigm, that reduces to the one-to-many zero-knowledge proof-of-knowledge functionality $\mathcal{F}_{\mathrm{zkPoK}}^{1:\mathrm{M}}$, which can be compiled into its accountable counterpart through the aforementioned $\mathcal{ABC}$ transformation. This results in $\tau_{zk\text{-}scr}$, an accountable extension of BCLOS, where any breach in secure emulation leads to externally verifiable proof-of-misbehavior. The generality of this approach makes it applicable to *all* protocols, including lightweight ones tailored to specific functionalities. Applying the $\tau_{zk\text{-}scr}$ compiler to a general-purpose AMPC protocol secure against up to $t_{\mathrm{ne}} = \lceil n/2 \rceil - 1$ non-equivocating faults [18], [23] yields a (t, t_{ne})-accountable AMPC protocol that: (i) ensures all MPC hyperproperties (1–4) when the number of malicious faults f does not exceed the nominal-case threshold t, with $t < n/3$, and (ii) guarantees publicly verifiable identification of at least $(n - 2t)$ faulty processes whenever (hyper)safety is violated and $f \in (t, t_{\mathrm{acc}}]$, with $t_{\mathrm{acc}} = t_{\mathrm{ne}}$. This naive approach is, however, unsatisfactory. First, in terms of feasibility, one would expect t_{acc} to exceed t_{ne}. Second, the construction relies on computationally heavy, general-purpose [40] zero-knowledge proofs of knowledge (zkPoK) applied to protocol state transitions. Third, every message must be reliably broadcast, incurring multiplicative overhead in both communication and latency. While tolerable for lightweight protocols, this overhead is prohibitive when applied to the already costly general AMPC protocols used for arbitrary functionalities.

Accountable AMPC. A natural next step is therefore to build an accountable, dual-threshold AMPC protocol: one that evaluates an arithmetic circuit and provides different guarantees depending on the actual number of corruptions. This is a tempting goal because efficient AMPC protocols already follow a Shamir-sharing offline/online paradigm, with communication essentially linear in the circuit size [41], and because high-threshold asynchronous complete (verifiable) secret sharing (HACSS) can in principle preserve privacy beyond the nominal threshold [42]. A first question is whether these protocols can simply be made accountable. For sharing, the answer is partly yes: one can adapt the HACSS protocol of [42] by having the dealer reliably broadcast a publicly verifiable encrypted vector containing one share for each process, encrypted under the corresponding public key, and by replacing reliable broadcast with accountable reliable broadcast. The high degree still gives high-threshold privacy, and disagreement on the advertised sharing object yields irrefutable detection. This baseline is, however, inherently quadratic, since all processes must receive enough information to verify the full encrypted share vector. It therefore does not recover the communication profile of recent efficient AMPC protocols, such as [41].

The second obstacle is multiplication. In the online phase, multiplications are usually handled through Beaver triples: preprocessed sharings $[a]_*^\ell$, $[b]_*^\ell$, and $[c]_*^\ell$ such that $c = a \cdot b$. Given such a triple, a multiplication gate $z = x \cdot y$ can be evaluated without increasing the sharing degree: parties open $A = x - a$ and $B = y - b$, then locally compute $[z]_*^\ell = [c]_*^\ell + A \cdot [b]_*^\ell + B \cdot [a]_*^\ell + AB$. Thus, if triples were available for free, the online protocol would be conceptually simple. Parties would share their inputs using accountable HACSS, agree through accountable agreement on a core set of $n - t$ correct sharings, and then evaluate the circuit over degree-ℓ sharing polynomials. Polynomial commitments and openings would support erasure-code-style reconstruction, both when opening the masked values needed for Beaver-assisted multiplication and when delivering each party's output wires. The difficulty is that Beaver triples do not come for free. Generating them directly would require multiplying degree-ℓ sharing polynomials, which produces degree-2ℓ objects. In asynchrony, only a quorum of $n-t$ processes can be relied on, and this is not enough to manipulate degree-2ℓ sharings while preserving privacy in the degraded regime. This degree-slack bottleneck is the main reason why the offline phase requires separate techniques.

Our contribution. We present the first accountable asynchronous MPC protocol for any $t_{\mathrm{acc}} < n-t$. Our construction follows the standard secret-sharing approach within the offline/online paradigm. Beaver triples [43] are generated during an offline phase, independent of the circuit $\mathcal{C}$ except for its size $|\mathcal{C}|$, and the online phase evaluates $\mathcal{C}$ using these

triples. A setup phase, fully independent of $\mathcal{C}$, generates the required cryptographic material.

All phases assume a computational security parameter κ, a finite field $\mathbb{F}$ of prime characteristic $p \geq 2^\kappa$, and a transparent setup (i.e., a bulletin-board public-key infrastructure and a uniform common random string). For MPC, we additionally consider an arithmetic circuit $\mathcal{C}$ over $\mathbb{F}$ with multiplicative depth D, $\Omega(n)$ multiplication gates per multiplicative layer, and a total of c_M multiplication gates. We note c_R the number of random wires, and c_E the number of external input/output wires.

Online Phase. The online phase relies on a new core primitive: the *accountable* additively homomorphic packed and batched high-threshold asynchronous complete (verifiable) secret-sharing functionality, denoted $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}$.

Theorem 1. *Let $t_{\text{acc}} < \ell \in [t, n-t)$ and $\tau = \ell$. Assume a powers-of-τ setup and the corresponding Strong Diffie–Hellman assumption. There exists a realization $\mathcal{P}_{\text{pbhacss}}^{\text{AH,acc}}$ of the functionality $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}$ with $O(Kn\kappa)$ communication complexity in both the sharing and retrieval phases, for batches of $K = \Omega(n)$ sharings, each packing $\ell - t_{\text{acc}} + 1$ secrets.*

This is both the first accountable asynchronous secret-sharing protocol and the first realization, (even without accountability) of $\mathcal{F}_{\text{pbhacss}}$ achieving linear communication.[2] At a high level, the protocol combines techniques from [44] and [42]. Our realization of $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}$ is designed to rely only on zero-knowledge proofs for linear relations, avoiding the need for non-interactivity or witness succinctness. This enables the use of efficient generic Ω-protocols [46], which can be made non-interactive [47]. A key technical point is handling the retrieval phase so that correctness violations are always detectable in degraded executions; this property cannot be obtained by a naive use of erasure codes.

From this result, standard techniques immediately yield an online MPC theorem given access to correlated randomness (Beaver triples):

Theorem 2. *Let $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}\rangle$ (resp., $\mathcal{F}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}\rangle$) be the accountable Beaver triple generation (resp., shared randomness) functionality, storing c_M Beaver triples (resp., c_R independent random secrets) in $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}$. There exists a realization of $\mathcal{F}_{\text{ampc}}^{\text{acc}}$ in the $(\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}, \mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}\rangle, \mathcal{F}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}\rangle, \mathcal{F}_{\text{idx-acs}}^{\text{acc}})$-hybrid model, evaluating $\mathcal{C}$ with $O(D)$ round complexity and $O\big(\kappa n(c_M + c_R + c_E)\big)$ communication complexity.*

Note that for $c_R = \Omega\big(n(n - t - t_{acc})\big)$, the realization of $\mathcal{F}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}\rangle$ in the $(\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}, \mathcal{F}_{\text{idx-acs}}^{\text{acc}})$-hybrid model can be achieved with $O\Big(\frac{\kappa n^2 c_R}{n-t-t_{acc}}\Big)$ communication . This follows from Theorem 1 combined with standard random extraction techniques [48]. In particular, when $n - t - t_{acc} \in \Omega(n)$ (e.g., $(t, t_{acc}) \simeq (\frac{n}{3}, \frac{7n}{12})$), this yields amortized linear communication per shared random number.

2. Related results exist for weaker notions of dual-threshold privacy [44], [45].

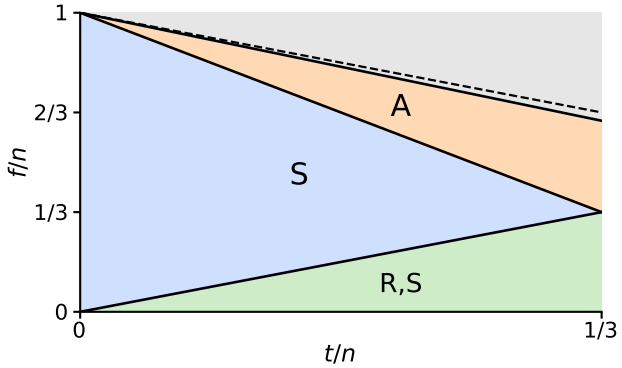


Figure 1. AAMPC's security regimes as a function of t/n, and f/n. R : Robustness (GOD); S : (Hyper)Safety; A : Accountability.

Offline Phase. We present a cubic offline protocol that serves three purposes. First, it generates Beaver triples for a certain realization of $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}$ at cubic communication cost. Second, these triples can be used directly to realize $\mathcal{F}_{\text{ampc}}^{\text{acc}}$, yielding an accountable AMPC protocol with overall cubic communication. In particular, this instantiation also produces the required SDH parameters for the polynomial commitment scheme used in the online phase. Third, via an accountable mask-opening procedure, the generated triples can be migrated to the $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}$ realization used in the online phase.

Theorem 3. *There exists a realization of the functionality $\mathcal{F}_{\text{ampc}}^{\text{acc}}$ that evaluates $\mathcal{C}$ with $O(\kappa n^3|\mathcal{C}|)$ communication complexity and $O(D)$ round complexity.*

This construction can be viewed as the accountable counterpart of the AMPC protocol of [18], originally presented in the non-equivocating model, but with a significantly simplified presentation.

Paper organization. Section 2 provides a high-level overview of our construction. Section 3 introduces the model, mainly AUC [49]. Section 4 presents our main technical result, Theorem 1, namely the linear Shamir-type realization of $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}$. Section 5 presents the online phase and states its security theorem. Appendix A recalls the circuit-evaluation conventions used by the online phase. Appendix B proves the online theorem. The remaining appendices give the transparent-setup offline phase, the full AUC definitions, and the auxiliary accountable functionalities.

2. Technical Overview

Notation. We use lowercase boldface letters for vectors and uppercase boldface letters for matrices (e.g., $\mathbf{v}$ and $\mathbf{M}$). Vector and matrix indices start from 1; we write $\mathbf{v}[i]$ for the i-th entry of a vector $\mathbf{v}$, and $\mathbf{M}[i, j]$ for the (i, j)-th entry of a matrix $\mathbf{M}$. We denote by $\mathbf{M}[i, :]$ (resp. $\mathbf{M}[:, j]$) the i-th row (resp. j-th column) of $\mathbf{M}$. For any index set $J \subset \mathbb{N}$,

we write $\mathbf{M}[J][:]$ to denote the submatrix of $\mathbf{M}$ obtained by restricting to the rows indexed by J; other submatrix restrictions are defined analogously.

Let p denote the prime characteristic of the field over which the arithmetic circuit is defined. All additive domains to which we apply explicit linear-algebraic operations are abelian groups of prime exponent p, and therefore naturally form vector spaces over $\mathbb{F}_p$. Scalar multiplication $\odot$ by elements of $\mathbb{F}_p$ is defined by repeated application of the group operation $\oplus$.

Interpolation and coding over algebraic structures. Throughout our protocols, reconstruction of secrets from shares amounts to interpolation, which can equivalently be described as decoding a linear code. This unified viewpoint is also the basis for our batching techniques.

Vandermonde Encoding. Let $(G, \oplus)$ be an abelian group of prime exponent p. We fix distinct evaluation points $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_p^n$.

For an integer $K \leq n$, define the Vandermonde matrix

$$\mathbf{V}_{\boldsymbol{\alpha}}^K[j,k] := \alpha_j^{k-1} \in \mathbb{F}_p, \qquad \mathbf{V}_{\boldsymbol{\alpha}}^K \in \mathbb{F}_p^{n \times K}.$$

For any subset $J \subseteq [n]$ of size K, the submatrix $\mathbf{V}_{\boldsymbol{\alpha}}^K[J][:]$ is invertible over $\mathbb{F}_p$; write $\boldsymbol{\Lambda}_J^{\boldsymbol{\alpha}} := (\mathbf{V}_{\boldsymbol{\alpha}}^K[J][:])^{-1}$.

- **Encoding (evaluation).** For $\mathbf{x} \in G^K$, define

$$\mathsf{encode}_{\boldsymbol{\alpha}}(\mathbf{x}) = \mathbf{V}_{\boldsymbol{\alpha}}^K \circledast \mathbf{x} \in G^n,$$
$$(\mathbf{V}_{\boldsymbol{\alpha}}^K \circledast \mathbf{x})[j] = \bigoplus_{k=1}^{K} \mathbf{V}_{\boldsymbol{\alpha}}^K[j,k] \odot \mathbf{x}[k].$$

  This is polynomial evaluation at $\alpha_1, \ldots, \alpha_n$ when $\mathbf{x}$ is the coefficient vector of a polynomial of degree $K-1$. We also write $\mathsf{val} \triangleq \mathsf{encode}$.
- **Decoding (interpolation).** For $J \subseteq [n]$ and $\mathbf{y} \in G^{|J|}$, define

$$\mathsf{decode}_{(\boldsymbol{\alpha},J)}(\mathbf{y}) = \boldsymbol{\Lambda}_J^{\boldsymbol{\alpha}} \circledast \mathbf{y} \in G^{|J|}.$$

  Thus, if $\mathbf{y} = \mathsf{encode}_{\boldsymbol{\alpha}}(\mathbf{x})$ for $\mathbf{x} \in G^{|J|}$, then $\mathsf{decode}_{(\boldsymbol{\alpha},J)}(\mathbf{y}_J) = \boldsymbol{\Lambda}_J \circledast (\boldsymbol{\Lambda}_J^{-1} \circledast \mathbf{x})$, where $\mathbf{y}_J$ is the restriction of $\mathbf{y}$ to coordinates in J. Since G is a $\mathbb{F}_p$-vector space, $\boldsymbol{\Lambda}_J \circledast (\boldsymbol{\Lambda}_J^{-1} \circledast \mathbf{x}) = (\boldsymbol{\Lambda}_J \boldsymbol{\Lambda}_J^{-1}) \circledast \mathbf{x} = \mathbf{x}$. In particular, the Shamir secret of $\mathbf{x}$ is

$$\mathsf{interpol}_{(\boldsymbol{\alpha},J)}(\mathbf{y}_J) \;=\; \mathsf{decode}_{(\boldsymbol{\alpha},J)}(\mathbf{y}_J)[1],$$

  the first coefficient of the interpolated polynomial.
- **Random extraction.** Let $\mathbf{S} \in R^{K \times (n-t)}$ be a batch matrix whose columns include $n - t - t_{\mathsf{acc}}$ uniformly random ones over $\mathbb{F}_p$. Multiplying on the right by a hyperinvertible Vandermonde matrix $\mathbf{M} \in \mathbb{F}_p^{(n-t) \times (n-t-t_{\mathsf{acc}})}$ yields

$$\mathbf{R} \triangleq \mathsf{ExtractRand}(\mathbf{S}) = \mathbf{S} \odot \mathbf{M} \in R^{K \times (n-t-t_{\mathsf{acc}})}.$$

  which consists of $K(n - t - t_{\mathsf{acc}})$ uniformly random linear combinations.
- **Extension.** We extend $\mathsf{encode}, \mathsf{decode}, \mathsf{val}, \mathsf{interpol}$ to matrix arguments (with notations $\mathsf{Encode}, \mathsf{Decode}, \mathsf{Val}, \mathsf{Interpol}$) by applying multiplication with $\mathbf{V}^K$ or $\boldsymbol{\Lambda}_J$ rowwise or columnwise in the same way.
  When using additively homomorphic encryption with plaintext of prime exponent p, the above $\mathbb{F}_p$-linear operations extend to ciphertexts via homomorphic evaluation, even though the ciphertext space itself need not have exponent p.

Evaluating an Arithmetic Circuit Through Secret Sharing. We consider functionalities represented as arithmetic circuits $\mathcal{C}$ over a finite field $(\mathbb{F}, \oplus_{\mathbb{F}}, \otimes_{\mathbb{F}})$ of prime characteristic p, where p itself is cryptographically large, i.e., $\log_2 p \geq \kappa$, and κ denotes the computational security parameter. For simplicity, we present the protocol over the prime field $\mathbb{F} = \mathbb{F}_p$, although our $\mathbb{F}_p$-linear constructions extend naturally to fields $\mathbb{F}_{p^{k>1}}$.

Each gate performs either addition $\oplus_{\mathbb{F}}$ or multiplication $\otimes_{\mathbb{F}}$. The circuit operates on input wires (corresponding to participants' inputs) and produces output wires (corresponding to participants' outputs). If privacy is ignored, processes can simply use an (indexed) Agreement on a Core Set functionality $\mathcal{F}_{\text{idx-acs}}$ to agree on $n - t$ inputs and then evaluate $\mathcal{C}$ step by step. To ensure privacy, however, $\mathcal{F}_{\text{idx-acs}}$ is instead used to agree on a core set of parties that correctly *share* their inputs via an Asynchronous Complete (Verifiable) Secret Sharing functionality $\mathcal{F}_{\text{acss}}$. Computation is then carried out on shares rather than on plaintext values, and outputs are privately reconstructed at the end.

A secret $s \in \mathbb{F}$ is said to be shared with reconstruction threshold $\ell \in [t, n-t)$ if each process p_i holds a *share* $[s]_i^\ell \in \mathbb{S}$ such that: (1) there exists a t-resilient reconstruction protocol that allows correct processes to *open* s, and (2) no coalition of up to ℓ malicious parties learns anything about s unless reconstruction is initiated by at least one correct process.

This is typically realized à la Shamir: each process p_i holds the evaluation of a degree-ℓ polynomial $\phi_s \in \mathbb{F}_\ell[X]$ at some public point $\alpha_i \in \mathbb{F}_p$. Reconstruction proceeds by exchanging evaluations and interpolating ϕ_s to recover $s = \phi_s(0)$. When $f \leq t = \ell$, this can be done with perfect security via error correction; when $\ell \in [t, n-t)$, or when safety violation must be detected if $f > t$, one may leverage additional information built on top of a (computationally secure) polynomial commitment scheme PC. The sharing $[\cdot] : (\mathbb{F}, \oplus_{\mathbb{F}}) \to (S, \oplus_{\mathbb{S}})$ is typically $\mathbb{F}_p$-linear. Thus addition and multiplication by a scalar in $\mathbb{F}_p$ can be performed locally. This is the case of the Shamir secret sharing over $\mathbb{F}[X]$. However, Shamir secret sharing is not multiplicatively homomorphic, since multiplying two degree-ℓ polynomials yields a degree-2ℓ polynomial. Dedicated techniques are therefore required for multiplication.

Offline/Online Paradigm. To avoid costly interactive subprotocols, multiplications are performed using preprocessed *Beaver triples* $([a]_*^\ell, [b]_*^\ell, [c = a \otimes_{\mathbb{F}} b]_*^\ell)$ generated during an offline phase. This preprocessing is independent of the function to be evaluated, except for the number of multiplication gates required, and can be efficiently batched.

During the online phase, suppose parties hold $[x]^\ell_*$ and $[y]^\ell_*$ and want to compute $[z]^\ell_*$ with $z = x \otimes_{\mathbb{F}} y$. They proceed as follows:

1) Open $[A]^\ell_* = [x \oplus_{\mathbb{F}} (-a)]^\ell_*$ and $[B]^\ell_* = [y \oplus_{\mathbb{F}} (-b)]^\ell_*$, and locally compute $[A \otimes_{\mathbb{F}} B]^\ell_*$
2) Locally compute $[z]^\ell_* = [A \otimes_{\mathbb{F}} B]^\ell_* \oplus_{\mathbb{S}} (A \odot_{\mathbb{S}} [b]^\ell_*) \oplus_{\mathbb{S}} (B \odot_{\mathbb{S}} [a]^\ell_*) \oplus_{\mathbb{S}} [c]^\ell_*$, which equals $[x \otimes_{\mathbb{F}} y]^\ell_*$ by the homomorphic properties of the sharing scheme.

Computing (resp., accountably computing) the first two components of a batch of high/dual-threshold Beaver triples relies on the (accountable) high/dual-threshold shared randomness functionality $\mathcal{F}_{\text{sr}}$ (resp., $\mathcal{F}^{\text{acc}}_{\text{sr}}$), which in turn builds on $\mathcal{F}^{\text{AH}}_{\text{bhacss}}$ (resp., $\mathcal{F}^{\text{AH,acc}}_{\text{bhacss}}$). A realization of $\mathcal{F}^{\text{AH}}_{\text{bhacss}}$ consists of a pair of protocols $(\mathsf{Sh}, \mathsf{Rec})$ that allow a dealer to share a batch of secrets such that each process obtains its designated shares with the required dual-threshold privacy guarantees.

The protocol satisfies (among others) two key properties: (i) *completeness*: if a correct process acknowledges receipt of its batch, then all correct processes will eventually acknowledge receipt of theirs; and (ii) *binding*: once the first acknowledgement occurs, there exists a unique batch of secrets that all correct processes can later reconstruct. In its accountable variant, if $f > t$ and binding is violated, the protocol guarantees externally verifiable detection of at least $n - 2t$ malicious parties.

Given a realization of $\mathcal{F}^{\text{AH,acc}}_{\text{bhacss}}$ with (amortized) linear communication per secret (see later), one can directly construct a $\mathcal{F}^{\text{acc}}_{\text{sr}}\langle\mathcal{F}^{\text{AH,acc}}_{\text{bhacss}}\rangle$'s realization with (amortized) $O(\frac{\kappa n^2}{(n-t-t_{acc})})$ communication per (uniformly sampled) random shared secret. Each process acts as a dealer and shares a K-batch of secrets through $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$. Upon acknowledgement of $(n-t)$ correct sharings, processes invoke $\mathcal{F}^{\text{acc}}_{\text{idx-acs}}$ to agree on a core set of $n - t$ dealers whose secrets were correctly shared. Completeness ensures that all correct processes eventually obtain their shares, while binding guarantees that maliciously chosen secrets are independent of those generated by honest parties.

Thanks to additive homomorphism, any linear combination of shared secrets that includes at least one honestly generated uniform random value yields another uniform random shared secret. Thus, the shared material can be arranged into a matrix $\mathbf{S} \in \mathbb{S}^{K\times(n-t)}$. Rather than using $\mathbf{S}$ naively, we apply random extraction [48]: multiplying $\mathbf{S}$ by a hyperinvertible matrix $\mathbf{M}$ of dimension $\big((n-t)\times(n-t-t_{\text{acc}})\big)$ (e.g., a Vandermonde matrix) yields $\mathbf{R} \in \mathbb{S}^{K\times(n-t-t_{\text{acc}})}$, which provides $K \cdot (n - t - t_{\text{acc}})$ uniformly random shared secrets from independent linear combinations.

If the parameters are chosen so that $2c_M + c_R = K(n - t - t_{\text{acc}})$, then the matrix $\mathbf{R}$ suffices to provide all c_R random wires and the first two components of the c_M Beaver triples. The challenge is to complete these triples.

Multiplication and the degree slack problem. Classical re-sharing (also known as degree-reduction) techniques, such as those of [50], become infeasible in our asynchronous setting when $\ell > t$, since they require manipulating degree-2ℓ polynomials. Reconstructing such polynomials needs at least $2\ell + 1$ evaluations, but correct processes can obtain only $n - t = 2t + 1$ evaluations when $n = 3t + 1$, which is insufficient as soon as $\ell > t$. In synchronous settings, this obstacle can be avoided: with $\ell = t$ and $n > 2t$, enough evaluations are available, and uncooperative processes can be excluded or emulated. Similarly, in classical asynchronous MPC with $\ell = t$, the bound $n - t = 2\ell + 1$ still suffices. With $\ell > t$, however, the slack between $n - t$ and 2ℓ disappears. This bottleneck, which we call the *degree slack problem*, cannot be bypassed by trivially sharing auxiliary points for reconstruction. For instance, if evaluations $\phi^a(\beta_j)$ and $\phi^b(\beta_j)$ are themselves shared via polynomials ψ^a_j, ψ^b_j of degree ℓ, retrieving the product share $\phi^a\phi^b(\beta_j)$ requires reconstructing the degree-2ℓ product polynomial $\psi^a_j\psi^b_j$. This auxiliary reconstruction remains trapped by the same $2\ell + 1 > n - t$ barrier. Attempts to artificially increase evaluations (e.g., each process providing two) would double the information revealed, reducing the privacy threshold to $\ell/2$ and undermining security. Multiplication thus becomes rigid: raising the degree breaks liveness, while lowering it breaks privacy.

The Franklin-Haber Method. It was observed in [51] (see also [52], [53]) that a Beaver triple can be easily computed if the sharing process is staged. Specifically:

1) Processes first establish a shared random value $[a]$.
2) Subsequently, processes generate two shared values $[b] = \bigoplus^{i\in Q}_{\mathbb{S}}[b_i]$ and $[c] = \bigoplus^{i\in Q}_{\mathbb{S}}[c_i]$ for an authorized set $Q \subset [n]$, where each b_i is a random contribution from p_i, and each $[c_i] = b_i \odot_{\mathbb{S}} [a] = [a \otimes_{\mathbb{F}} b_i]$.

This approach bypasses the degree slack. Instead, it requires each p_i to compute shares $[c_i]_j = b_i \odot [a]_j$ for all $j \in [n]$. To ensure the privacy of $[a]_j$ is not jeopardized, one can utilize additively homomorphic encryption $\big(\text{AHE}\big)$ [54] to perform these scalar multiplications in the encrypted domain.

An AHE scheme $\big(\mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}\big)$ with plaintext space $\mathsf{PT} \simeq \mathbb{F}_p$ supports correct homomorphic evaluation of linear arithmetic circuits over PT. For any arithmetic circuit $\mathcal{C}$ of multiplicative depth 0, and any messages $m_1, \ldots, m_k \in \mathsf{PT}$, we have $\mathsf{Dec}\big(\mathsf{Eval}(\mathcal{C}, \mathsf{Enc}(m_1), \ldots, \mathsf{Enc}(m_k))\big) = \mathcal{C}(m_1, \ldots, m_k)$.

For convenience, we overload the symbols $\oplus_{\mathsf{CT}}$ and $\odot_{\mathsf{CT}}$ on ciphertexts to denote the result of homomorphic evaluation of the corresponding addition and scalar-multiplication circuits. These operations do not endow the ciphertext space CT with an intrinsic $\mathbb{F}_p$-linear or vector-space structure.

The Backes-Bendun-Choudhury-Kate method In the terminology of [18], a process p_i is a *privileged party of sharing of* s, if it holds $\{\mathsf{Enc}_{pk_j}([s]^\ell_j, *)\}_{j\in[n]}$. When both the encryption and the sharing are additively homomorphic, a privileged party of all sharings selected by $\mathcal{F}^{\text{acc}}_{\text{idx-acs}}$ can linearly combine the encrypted shares to produce encryptions of new sharings. Concretely, suppose a random sharing $[a]^\ell_*$ is generated , so that all processes act as privileged parties of sharing a. Each party p_i then shares a fresh random contribution $[b_i]^\ell_*$ and, in parallel, produces an indirect sharing of $[a \otimes_{\mathbb{F}} b_i]^\ell_*$ by homomorphically combining the

encrypted shares of a with its own value b_i plus masking noise, where CT denotes ciphertext space. Typically, p_i computes $\{\big(b_i \odot_{CT} \mathsf{Enc}_{pk_j}([a]^\ell_j, r_j)\big) \oplus_{\mathsf{CT}} \mathsf{Enc}_{pk_j}(0, r'_{i,j})\}_{p_j \in \Psi} = \{\mathsf{Enc}_{pk_j}([a \otimes_{\mathbb{F}} b_i]^\ell_j, r''_{i,j})\}_{p_j \in \Psi}$. In this way, p_i never learns a, yet every recipient p_j can decrypt its designated ciphertext and obtain a valid share $[a \otimes_{\mathbb{F}} b_i]^\ell_j$. Once a second core set Q of valid dealers is established, these sharings can be aggregated into consistent global sharings $[b]^\ell_* = \bigoplus_{\mathsf{CT}}^{i \in Q} [b_i]^\ell_*$ and $[a \otimes_{\mathbb{F}} b]^\ell_* = \bigoplus_{\mathsf{CT}}^{i \in Q} [a \otimes_{\mathbb{F}} b_i]^\ell_*$, yielding a Beaver triple $([a]^\ell_*, [b]^\ell_*, [a \otimes_{\mathbb{F}} b]^\ell_*)$. The corresponding UC-realization is deferred to Appendix F.

Linear Shamir-based Secret Sharing. Our linear realization of $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}$ combines two prior approaches: (i) the quadratic $\mathcal{F}^{\mathrm{acc}}_{\mathrm{hacss}}$ realization of [42], which builds on *verifiable encryption* (VE), and (ii) the linear-cost $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bacss}}$ realization of [55], based on the *encrypt-and-disperse* paradigm. A setup-free linear $\mathcal{F}_{\mathrm{bacss}}$'s realization with dual-threshold reconstruction was proposed in [45]. However, their scheme does not achieve dual-threshold privacy: although the secrecy threshold satisfies $\ell > t$ once the protocol terminates, during the execution the effective secrecy threshold is only t. The following description is not the exact algorithm used in our protocol, but rather a streamlined variant meant to highlight the key ingredients and convey the intuition. The final construction, presented in Section 4, refines these steps to ensure efficiency, and full UC-security guarantees.

Additively homomorphic polynomial commitment. We assume the availability of a $\mathbb{F}_p$-compatible polynomial commitment scheme $\mathsf{PC} = (\mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ for polynomials over $\mathbb{F}[X]$. Compatibility means that commitment and opening objects live in groups $\mathbb{H}_c$ and $\mathbb{H}_\theta$ are groups of order p, i.e. $\mathbb{F}_p$-vector space, while $\mathsf{Com}, \mathsf{Open}, \mathsf{Ver}$ are $\mathbb{F}_p$-linear (e.g., KZG commitments [56]). The scheme is $\mathbb{F}_p$-linear in the sense that commitments to linear combinations of polynomials can be derived by applying the corresponding linear operations directly in the commitment space, and similarly for evaluation proofs. Correctness of the commitment scheme guarantees that verification enforces consistency with the underlying polynomial semantics over $\mathbb{F}_p$.

Additively homomorphic encryption. We assume an $\mathbb{F}_p$-compatible additively homomorphic encryption scheme $\mathsf{AHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ whose plaintext space PT is a cyclic group of prime order p, and hence an $\mathbb{F}_p$-vector space (e.g., encoding-free ElGamal [54]). Concretely, PT is instantiated as a prime-order group with $\mathsf{PT} \cong (\mathbb{F}_p, \oplus_{\mathbb{F}_p})$, and encryption and decryption operate on group elements. Ciphertexts live in a group CT of size $O(|\mathsf{PT}|)$. The scheme is $\mathbb{F}_p$-linear in the sense that Enc is homomorphic with respect to the group operation on PT: given encryptions of plaintexts $m_1, m_2 \in \mathsf{PT}$, one can efficiently derive an encryption of $m_1 \oplus_{\mathsf{PT}} m_2$ by applying the corresponding group operation in CT, and, for any scalar $a \in \mathbb{F}_p$, derive an encryption of $a \odot_{\mathsf{PT}} m_1$ by applying the corresponding scalar multiplication in the ciphertext space.

Sharing phase. Consider a dealer wishing to distribute a batch $\mathbf{s} = [s_1, \ldots, s_K] \in \mathbb{F}^K$, with $K = t + 1$. The dealer samples uniformly random polynomials $\phi_{s_1}, \ldots, \phi_{s_K} \in \mathbb{F}[X]$ such that, for all $k \in [K]$, $\phi_{s_k}(0) = s_k$. For each $k \in [K]$, the dealer computes a commitment $c_k = \mathsf{PC.Com}(\phi_{s_k}) \in \mathbb{H}_c$ and reliably broadcasts $\mathbf{c} = [c_1, \ldots, c_K]$ via $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$.

The dealer then evaluates each polynomial at the public evaluation points $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_n]$ and forms the evaluation matrix $\boldsymbol{\Phi}$ with entries $\boldsymbol{\Phi}[k, j] = \phi_{s_k}(\alpha_j)$. For each pair $(k, j) \in [K] \times [n]$, the dealer computes a corresponding evaluation proof $\boldsymbol{\Theta}[k, j] = \mathsf{PC.Open}_{\alpha_j}(c_k, \phi_{s_k}) \in \mathbb{H}_\theta$ such that $\mathsf{Ver}_{\alpha_j}\big(c_k, \boldsymbol{\Phi}[k, j], \boldsymbol{\Theta}[k, j]\big) = 1$. Finally, each pair $(\boldsymbol{\Phi}[k, j], \boldsymbol{\Theta}[k, j])$ is encrypted under the public key pk_j of process p_j, yielding $\forall (k, j) \in [K] \times [n]$:

$$(\hat{\boldsymbol{\Phi}}[k, j], \hat{\boldsymbol{\Theta}}[k, j]) = \big(\mathsf{Enc}_{\mathsf{pk}_j}(\boldsymbol{\Phi}[k, j]; *), \mathsf{Enc}_{\mathsf{pk}_j}(\boldsymbol{\Theta}[k, j]; *)\big).$$

The dealer encodes $(\hat{\boldsymbol{\Phi}}^*, \hat{\boldsymbol{\Theta}}^*) = (\mathsf{Encode}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{\Phi}}), \mathsf{Encode}_{\boldsymbol{\alpha}}(\hat{\boldsymbol{\Theta}}))$.

For each p_i, the dealer sends the vectors $\hat{\boldsymbol{\Phi}}^*[i, :]$ and $\hat{\boldsymbol{\Theta}}^*[i, :]$, together with a zero-knowledge proof π_i attesting that these are consistent with $\mathbf{c}$.

Upon receiving $\mathbf{c}$ via $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$ and a valid pair $(\hat{\boldsymbol{\Phi}}^*[i, :], \hat{\boldsymbol{\Theta}}^*[i, :])$, process p_i responds with an acknowledgement for $\mathbf{c}$, signed using an $(n-t)$-threshold signature scheme. It also computes $\mathbf{c}^* = \mathsf{Encode}_{\boldsymbol{\alpha}}(\mathbf{c})$. Once the dealer collects $n-t$ such partial signatures, it combines them into a full threshold signature and reliably broadcasts it.

When $\mathbf{c}$ has been reliably delivered, each process p_i forwards $(\hat{\boldsymbol{\Phi}}^*[i, j], \hat{\boldsymbol{\Theta}}^*[i, j])$ to process p_j. Upon receiving a set I of at least K valid codewords ($K \geq n - 2t$), process p_j decodes

$$\hat{\boldsymbol{\Phi}}[:, j] = \mathsf{decode}_{(\boldsymbol{\alpha}, I)}(\hat{\boldsymbol{\Phi}}^*_I[:, j]))$$

$$\hat{\boldsymbol{\Theta}}[:, j] = \mathsf{decode}_{(\boldsymbol{\alpha}, I)}(\hat{\boldsymbol{\Theta}}^*_I[:, j])$$

which is possible because CT is equipped with $\mathbb{F}_p$-linear operations $(\oplus_{\mathsf{CT}}, \odot_{\mathsf{CT}})$. Then, p_j decrypts using sk_j to obtain $(\boldsymbol{\Phi}[:, j], \boldsymbol{\Theta}[:, j])$. To handle the degraded case ($f > t$), $\boldsymbol{\Phi}[:, j]$ is additionally checked against $\mathbf{c}$ and $\boldsymbol{\Theta}[:, j]$.

If the threshold-signed acknowledgement of $\mathbf{c}$ is reliably delivered, process p_j can safely acknowledge the sharing phase.

Correctness follows from the reliable delivery of the threshold-signed acknowledgement of $\mathbf{c}$, which guarantees that at least $n - t - f$ correct processes will forward their codewords, sufficient for recovery in the nominal case ($f \leq t$). Privacy comes from the encryption of shares, while correctness of the reconstructed batch is guaranteed by the soundness of the attached zero-knowledge proofs. In the degraded case ($f > t$), binding violation implies agreement violation in the $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$, which leads to detection.

Codeword Verification. The remaining challenge is to ensure that codewords are valid. Naively checking against $\mathbf{c}$ and retrying could lead to an exponential number of attempts under malicious behavior (especially if the dealer colludes with faulty processes). Alternatively, one could attach to each pair $(\hat{\boldsymbol{\Phi}}^*[i, j], \hat{\boldsymbol{\Theta}}^*[i, j])$ a dedicated zkSNARK π^j_i, but this would preclude batching all these proofs into a single proof for the entire row

$(\hat{\mathbf{\Phi}}^*[i,:], \hat{\mathbf{\Theta}}^*[i,:])$. Instead, we leverage the additive homomorphic properties of both encryption schemes and commitment schemes. For additively homomorphic encryption, $\mathbf{\Phi}^*[i,j] = \mathsf{Dec}_{\mathsf{sk}_j}(\hat{\mathbf{\Phi}}^*[i,j])$, $\mathbf{\Theta}^*[i,j] = \mathsf{Dec}_{\mathsf{sk}_j}(\hat{\mathbf{\Theta}}^*[i,j])$ so process p_j can directly recover $(\mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j])$. For additively homomorphic commitments, if for each $(k,j) \in [K] \times [n]$ the tuple $(\mathbf{\Phi}[k,j], \mathbf{\Theta}[k,j], \mathbf{c}(k))$ is a consistent opening proof, then for each $(i,j) \in [n]^2$ the same holds for $(\mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j], \mathbf{c}^*(k))$. Thus, process p_j can directly check the consistency of each chunk! Moreover, the opening proofs will be re-used in the reconstruction phase.

Reconstruction Phase. The linear opening protocol is comparatively simple. However, since the sharing degree is $\ell \in [t, n-t)$, classical online error-correction cannot be applied. Instead, shares $\mathbf{\Phi}[i,j]$ are accompanied by commitment openings $\mathbf{\Theta}[i,j]$ that guarantee their integrity before being dispersed, following the same structure as the other batched protocols. That is why dispersing $\mathbf{\Theta}$ is (doubly) crucial. Otherwise, upon receiving n candidate polynomial evaluations containing up to t errors, a process would need to brute-force among all $\binom{n}{n-t}$ possible subsets of $n-t$ evaluations, decoding each and checking it against the corresponding polynomial commitment. This would result in exponential computational overhead. To obtain amortized linear communication per reconstruction, the commitment scheme must support an additively homomorphic structure (e.g., KZG [56]), enabling the use of linear erasure codes.

The batched reconstruction follows a similar, though leaderless, pattern as the sharing phase. Suppose B secrets are shared via degree-ℓ Shamir polynomials, whose coefficients are collected into a matrix $\mathbf{\Xi} \in \mathbb{F}^{B \times (\ell+1)}$. The corresponding evaluation matrix is $\mathbf{\Phi} = \mathsf{Val}_{\boldsymbol{\alpha}}(\mathbf{\Xi})$, so that each process p_j locally holds the column $\mathbf{\Phi}[:,j]$, i.e., all polynomial evaluations at α_j, together with the associated opening proofs $\mathbf{\Theta}[:,j]$ and the common commitment vector $\mathbf{c}[:]$.

At a higher level, each process encodes its local evaluations via $\mathsf{encode}_{\boldsymbol{\alpha}}$, yielding the $n \times n$ matrix $\mathbf{\Phi}^*$, and similarly $\mathbf{\Xi}^*$, $\mathbf{\Theta}^*$, and (n vector) $\mathbf{c}^*$. Rather than sending each raw symbol $\mathbf{\Phi}[i,j]$, process p_j transmits the encoded pair $(\mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j])$ to each p_i, who verifies it against the corresponding commitment $c^*[i]$.

Once a process p_i has collected $\ell+1$ valid evaluations, it can reconstruct its row $\mathbf{\Xi}^*[i,:]$, and in particular broadcast $\mathbf{\Xi}^*[i,1]$, the component corresponding to the secret. Finally, upon receiving a set $\{\mathbf{\Xi}^*[i,1]\}_{i \in I}$ for $|I| = K$, any process can decode to retrieve $\mathbf{\Xi}[:,1] = \mathsf{Decode}_{(\boldsymbol{\alpha},I)}(\mathbf{\Xi}^*_I[:,1])$.

We address the degraded regime $f > t$ by enabling verification of the values $\{\mathbf{\Xi}^*[i,1]\}_{i \in I}$. For communication efficiency, a process p_i cannot transmit the entire vector $\mathbf{\Xi}^*[i,:]$, and the commitment $c^*[i]$ alone does not suffice to verify the correctness of $\mathbf{\Xi}^*[i,1]$. We therefore extend the sharing scheme so that a dealer sharing a value s additionally publishes a commitment $c_0 = \mathsf{Com}_0(s; r_0)$. The commitment scheme must satisfy the following properties. First, since the sharing mechanism is also used for inputs to the arithmetic circuit being evaluated, which are not necessarily uniformly random, the commitment must be hiding with respect to the randomizer r_0. Since this randomizer must itself be shared in parallel with s, exactly as described above, the commitment Com_0 is required to be *doubly* $\mathbb{F}_p$-linear, that is, $\mathbb{F}_p$-linear both in the committed value and in the randomizer. Finally, as the scheme is used to share inputs to the $\mathcal{F}^{\mathrm{acc}}_{\mathrm{ampc}}$ functionality, the resulting commitments must be equivocal. All these requirements are met by the equivocal Pedersen-like commitment scheme of [57]. Similarly, the compact additively homomorphic commitment to the sharing polynomial can be instantiated using a KZG-like polynomial commitment, where the associated SDH setup parameters are used as a trapdoor by the simulator to explain, *a posteriori*, a dummy commitment generated without knowledge of the committed value. The trapdoor polynomial commitment is presented in Appendix G.

Masked opening. The Beaver triples produced in the offline phase cannot be directly injected into the $\mathcal{F}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ realization $\mathcal{P}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ used online. The reason is structural: shares in $\mathcal{P}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ are tied to KZG-like polynomial commitments, whereas the offline phase relies on Pedersen-type material, and these representations are not algebraically compatible. We resolve this mismatch via an accountable mask-opening protocol that migrates triples (or, more generally, sharings) from the offline representation to the instance of $\mathcal{P}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ used in the online phase. This separation cleanly accommodates alternative offline triple-generation mechanisms, including future improvement based on different cryptographic assumptions (e.g., threshold homomorphic encryption [58], [59]). The Beaver triple migration, from co-shared randomness, is presented in Appendix D and E.

3. Model & Preliminaries

Accountable Universal Composability (AUC). The Accountable Universal Composability (AUC) framework [49] extends UC (details are given in Appendix H) with explicit accountability guarantees. It builds on iUC [60], itself formulated in the IITM model [61]; UC is known to embed into IITM as well [62]. Within IITM/iUC, composition with joint or global state follows directly from the base composition theorems and does not require separate GUC or JUC extensions [63], [64]. A globally available resource is modeled simply by exposing enough I/O tapes so that the environment and all relevant protocols interact with the same instance.

In AUC, ideal functionalities additionally specify how violations of their security properties (e.g., agreement violations) must lead to the exposure of malicious parties. Each accountable functionality is associated with a common *public judge*, denoted Judge: a protocol that can be locally executed by any party, including external observers, and that outputs *verdicts* from *publicly verifiable evidence*. The framework requires three properties: (i) *fairness*, meaning honest parties are never falsely accused; (ii) *external verifiability*, meaning any third party can validate the evidence; and (iii) *individual accountability*, meaning verdicts are of the form $\mathsf{dis}(D)$, declaring all parties in D dishonest. For simplicity, we

describe a single globally accessible instance Judge, although any correct party may locally execute it.

AUC also introduces a meta-party, called the *supervisor*, whose role is to track violations of the *underlying assumptions* (such as synchrony), and to ensure that the real and ideal worlds remain consistent when such assumptions fail. In our setting, the only assumption tied to violations is the threshold condition $f \leq t$, which is already monitored in UC-like frameworks via standard corruption or status queries; accordingly, we do not make use of the supervisor.

Accountable functionalities extend the adversarial interface with explicit *Break* requests. We consider in particular BreakLiveness and BreakAgreement($\boldsymbol{\pi}$). A BreakLiveness request may affect liveness as soon as $f > t$, by allowing the adversary to indefinitely delay output notifications (except detection). An effective BreakAgreement request may affect safety only once $f \geq n-2t$, and only by providing a *judicial report* $\boldsymbol{\pi}$ that enables incrimination. Concretely, this means that for each correct party p_i, the functionality can deliver a proof $\boldsymbol{\pi}[i]$ such that submitting $\boldsymbol{\pi}[i]$ to the judge yields a verdict $\mathsf{dis}(D_i)$ with $|D_i| \geq n - 2t$. For technical completeness, a functionality may also disseminate self-incriminating proofs without affecting safety or liveness. This models *absurd self-incrimination*, where parties voluntarily reveal evidence against themselves without gaining any advantage, a behavior that can occur in the real world and must remain simulatable.

The generation and verification of judicial reports are handled by an auxiliary functionality $\mathcal{F}_{\mathrm{judi}}$, which can be viewed as the multi-signature analogue of the certification functionality $\mathcal{F}_{\mathrm{cert}}$ [65]. Accordingly, all accountable realizations are presented in the $\mathcal{J}$-hybrid model, where $\mathcal{J} = \mathsf{Judge} \mid \mathcal{F}_{\mathrm{judi}}$. This setup provides external verifiability while keeping the environment interface identical in both the ideal and real worlds. A complete specification of $\mathcal{J}$ is given in Appendix J.

At a high level, all accountability guarantees in our construction are inherited from easily accountable agreement functionalities (see below), whose concrete realization ultimately reduces to the confirmer functionality $\mathcal{F}_{\mathrm{conf}}^{\mathrm{acc}}$ [28]. All intermediate layers merely forward judicial reports and the resulting verdicts upward. We therefore omit explicit evidence-collection mechanisms at those layers, as they are already provided by the concrete realization of $\mathcal{F}_{\mathrm{conf}}^{\mathrm{acc}}$ in [28].

Liveness and fair executions. As in standard asynchronous MPC, liveness is stated only for fair executions [66]. At the concrete message-passing level, fairness means that messages sent between so-far honest parties are eventually delivered. At the ideal-functionality level, it means that delayed outputs whose preconditions are enabled are eventually released. Our simulation-based statements preserve fairness: a fair real execution is mapped to a fair ideal execution. Thus, when $f \leq t$, honest inputs are provided, and honest-to-honest messages are eventually delivered, the matched ideal execution cannot remain forever stuck before output. Conversely, when $f > t$ and the adversary causes a hang, the execution is matched to an ideal execution using BreakLiveness; no additional information is leaked by the loss of progress.

Processes, Network, and Adversary. We consider a static set of n processes $\Psi = \{p_1, \ldots, p_n\}$ with publicly known identities, communicating via message passing. Communication takes place over point-to-point channels that are reliable, private, and authenticated. Consequently, (i) every pair of processes can communicate, (ii) the recipient can always identify the sender, (iii) the adversary learns no information beyond the message length when a message is exchanged between two correct processes, and (iv) any message sent by one process to another is eventually delivered in fair executions. The network is fully asynchronous, with no a priori bound on message delays.

The adversary interacts both with the network and with the processes. It is polynomially bounded and assumed unable to break the cryptographic primitives used in our constructions (e.g., zero-knowledge proofs, encryption, signatures, or commitments). The adversary controls message scheduling, deciding when messages in transit are delivered, and may statically corrupt up to $f \in [0..n]$ processes.

We distinguish two fault thresholds. The *optimistic* threshold $t \in [0, n/3)$, and the *pessimistic* (accountability) threshold $t_{acc} \in (t,\, n - t)$, bounds the number of faults under which violations may occur but are guaranteed to be detectable.

Functionalities. Our protocols rely on several lower-level functionalities, presented in detail in Appendix I. We briefly summarize them here.

- $\mathcal{F}_{\mathrm{zkpok}}$ (Functionality 7) allows a prover to convince a verifier that it *knows* a witness w for a statement x such that (w, x) belongs to a given NP-relation, without revealing any information beyond the validity of the statement. In our construction, all NP-relations are algebraic and express homomorphism relations between abelian groups (e.g., linear relations over $\mathbb{F}_p$ or group-compatible commitments/encryptions). This structure enables the use of efficient Ω-protocols tailored to such linear relations [46], [47].
- **Easily accountable functionalities**. Their accountable variants explicitly model the case $f > t$, in which the adversary may violate the corresponding agreement property (possibly under additional restrictions, such as corrupting the dealer) by issuing a BreakAgreement request, at the cost of being eventually detected. These functionalities include:
  - $\mathcal{F}_{\mathrm{rbc}}$ (see Functionality 8): a reliable broadcast functionality allowing a dealer to disseminate a message. If an honest party reliably delivers the message, then all honest parties will eventually reliably deliver the same message.
  - $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M}}$ (see Functionality 10): a zkPoK variant of $\mathcal{F}_{\mathrm{rbc}}$, where a dealer proves to all parties that it knows a witness w for a statement x in a given NP-relation. If an honest party is convinced of x, then all honest parties will eventually be convinced of the same statement.
  - $\mathcal{F}_{\text{idx-acs}}$ (see Functionality 11): an indexed agreement on a core set, allowing processes

to agree on a set Q of size at least $n-t$ consisting of processes that correctly performed a given task (e.g., reliable broadcast, complete sharing).

- $\mathcal{F}_{\text{ack}}$ (see Functionality 9) allows processes to acknowledge a statement promoted by a given dealer. The statement can be a session ID, allowing processes to acknowledge the successful completion of the associated task driven by the dealer. If an honest process receives an acknowledgment, then all honest processes will eventually receive the same acknowledgment.

4. Accountable Packed and Batched High/Dual-Threshold Asynchronous Complete Secret Sharing (APBHACSS)

4.1. The $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}$ functionality

We give a high-level explanation of the functionality $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}$ (presented in Functionality 1), first ignoring all components related to metadata (shown in orange in the specification), and only later explaining their role and motivation.

Parameters and internal state (ignoring metadata). The functionality is parameterized by the number of parties n, a corruption threshold t, packing and batching parameters (P, K), an accountability threshold d, and two reconstruction thresholds $\ell_s \leq \ell_r$ (line 1). It maintains a table $Secrets$ storing, for each party, each packed row, and each secret identifier, the local share held by that party, as well as a reconstruction-initiator set $I[\mathsf{sid}]$ tracking which parties have requested reconstruction of a given secret identifier (line 4).

Sharing and agreement. Upon a sharing request $(\mathsf{Sh.share}, \mathsf{sid}_D, \mathbf{S})$ from a dealer D, the functionality installs identical packed shares of the batch $\mathbf{S}$ for all parties and all packed coordinates (line 16). This models perfect agreement and binding at the abstraction level: unless the adversary later triggers a break, all honest parties are guaranteed to hold consistent shares corresponding to the same underlying secrets.

Reconstruction initiation. Parties may initiate reconstruction of selected secret identifiers by issuing $(\mathsf{Rec.enable}, [\mathsf{sid}_1, \ldots, \mathsf{sid}_{K'}])$. For each identifier sid, the functionality records which parties have requested reconstruction, provided they locally hold a valid share for that identifier (line 22).

Dual-threshold reconstruction semantics. The functionality distinguishes two reconstruction thresholds. Once sufficiently many correct parties have requested reconstruction (at least $\ell_s + 1 - f$), the functionality leaks the corresponding local packed row to the simulator, modeling controlled secrecy leakage beyond the secrecy threshold (line 25). Once the stronger threshold $\ell_r + 1$ is reached, the functionality delivers the packed row to all parties who requested reconstruction, ensuring completeness (line 27).

Local linear operations and multiplication. The functionality supports local linear combinations of existing sharings via $(\mathsf{local_add}, \cdot)$, which are applied independently to each party's local shares and preserve the maximum algebraic level of the inputs (line 29).

Breaks and accountability. When the adversary controls more than t parties, it may force liveness violations. A BreakCompleteness request allows the simulator to indefi-

---

Functionality 1 $\mathcal{F}_{\text{pbhacss}}^{\text{AH,acc}}(\mathsf{Mt})$ (Accountable Packed & Batched HACSS)

1: **Parameters:**
2: $n, t, K, P, d \in \mathbb{N}$; field $(\mathbb{F}, \oplus_{\mathbb{F}}, \otimes_{\mathbb{F}})$ of prime exponent p; $\ell_s, \ell_r \in [t, n-t)$
3: $\mathsf{Mt} : \mathsf{Rand}_{\mathsf{Mt}}^{K} \times \mathsf{Secret}^{K} \to \mathsf{String}$ ▷ (additively homomorphic) metadata

4: **Variables:**
5: $\mathsf{Map}(\mathsf{SID} \times [\mathsf{P}] \times \mathsf{PID} \to \mathbb{F})$ $Secrets[][][]$ (init $\perp$)
6: $\mathsf{Map}(\mathsf{SID} \to \mathsf{ProcessSet})$ $I[]$ (init $\emptyset$) ▷ Processes who initiated reconstruction
7: $\mathsf{Map}(\mathsf{SID} \times \mathsf{PID} \to \mathsf{String})$ $meta[][]$

8: **Events / Interface:**
9: *request* $(\mathsf{Sh.share}, \mathsf{sid}_D, \mathbf{S})$: Sharing a batch of packed secrets $\mathbf{S} \in \mathsf{Mat}_{(P,K)}(\mathbb{F})$
10: *notification* $(\mathsf{Sh.receive}, \mathsf{sid}_D, r, m)$: Acknowledging the sharing driven by dealer D.
11: *request* $(\mathsf{Rec.enable}, \mathbf{sid} \in \mathsf{SSID}^{K'})$: Initiating batched&packed reconstruction.
12: *notification* $(\mathsf{Rec.retrieve}, \mathbf{sid}, \mathbf{S})$: Retrieving the secrets $\mathbf{S} \in \mathsf{Mat}_{(P,K')}(\mathbb{F})$.
13: *notification* $(\mathsf{detect}, \mathsf{sid}, G, \pi^{pom})$: Getting a proof π^{pom} incriminating culprits G
14: *request* $(\mathsf{local_add}, \mathsf{sid}_{sum}, [\mathsf{sid}_1, \ldots, \mathsf{sid}_m], [c_1, \ldots, c_m])$: Summing shares

15: **Functionality:**
16: **Upon receiving** $(\mathsf{Sh.share}, \mathsf{sid}_D, \mathbf{S} \in \mathsf{Mat}_{(P,K)}(\mathbb{F}))$ from dealer D:
17: **For all** $(i, q, k) \in [n] \times [P] \times [K]$:
18: $Secrets[\mathsf{sid}_D \circ k][q][i] \leftarrow \mathbf{S}[q, k]$; $Level[\mathsf{sid}_D \circ k] \leftarrow 0$ ▷ Binding/Agreement
19: $\mathbf{r}_D \leftarrow\!\!\$\ \mathsf{Rand}_{\mathsf{Mt}}^{K}$; $m = \mathsf{Mt}(\mathbf{S}, \mathbf{r}_D)$. **For all** $i \in [n]$: $meta[\mathsf{sid}_D][i] \leftarrow m$
20: **Send** $(\mathsf{Sh.receive}, \mathsf{sid}_D, \mathbf{r}_D, m)$ to D
21: **Send** $(\mathsf{Sh.receive}, \mathsf{sid}_D, \perp, meta[\mathsf{sid}_D][i])$ to each $p_i \neq D$ as a delayed output.

22: **Upon receiving** $(\mathsf{Rec.enable}, [\mathsf{sid}_1, \ldots, \mathsf{sid}_{K'}])$ from $p_i \in \Psi$:
23: **For each** $\mathsf{sid} \in \{\mathsf{sid}_1, \ldots, \mathsf{sid}_{K'}\}$: **If** $\forall q \in [P]$, $Secrets[\mathsf{sid}][q][i] \neq \perp$ **then**
24: $I[\mathsf{sid}] \leftarrow I[\mathsf{sid}] \cup \{p_i\}$.

25: **Upon** $|I[\mathsf{sid}] \cap \mathsf{Correct}| \geq \ell_s + 1 - f$: ▷ Dual-threshold secrecy leakage
26: **Send** $(\mathsf{Rec.retrieve}, \mathsf{sid}, Secrets[\mathsf{sid}][:][:])$ to Sim.

27: **Upon** $|I[\mathsf{sid}] \cap \mathsf{Correct}| = \ell_r + 1$: ▷ Completeness (packed row as a whole)
28: **Send** $(\mathsf{Rec.retrieve}, \mathsf{sid}, Secrets[\mathsf{sid}][:][i])$ to each $p_i \in I[\mathsf{sid}]$ as a delayed output.

29: **Upon receiving** $(\mathsf{local_add}, \mathsf{sid}_{sum}, [\mathsf{sid}_1, \ldots, \mathsf{sid}_m], [c_1, \ldots, c_m] \in \mathbb{F}^m)$ from p_i:
30: **For all** $q \in [P]$: $Secrets[\mathsf{sid}_{sum}][q][i] \leftarrow \bigoplus_{\mathbb{F}}^{j \in [m]} (c_j \odot_{\mathbb{F}} Secrets[\mathsf{sid}_j][q][i])$

31: **Upon receiving** $(\mathsf{BreakCompleteness})$ from Sim with $f > t$:
32: any notification can be indefinitely delayed.

33: **Upon receiving** $(\mathsf{BreakAgreement}, \mathsf{sid}_D, \underline{\mathbf{S}}, \mathbf{R}, \boldsymbol{\pi})$ from Sim with D corrupted:
34: **If** $f \geq n - 2t$ and $\forall i \in [n]$, the judge returns $\mathsf{dis}(G_i)$ on top of $\boldsymbol{\pi}[i]$ with $|G_i| \geq d$:
35: **Send** $(\mathsf{detect}, \mathsf{sid}_D, G_i, \boldsymbol{\pi}[i])$ to each p_i as a delayed output.
36: $\forall (i, q, k) \in [n] \times [P] \times [K]$: $Secrets[\mathsf{sid}_D \circ k][q][i] \leftarrow \underline{\mathbf{S}}[i, q, k]$
37: $\forall i \in [n]$: $meta[\mathsf{sid}_D][i] \leftarrow \mathsf{Mt}(\underline{\mathbf{S}}[i, :, :], \mathbf{R}[i, :])$

nitely delay pending outputs (line 31). Once the quorum-intersection bound permits conflicting views, namely for $f \geq n - 2t$, a BreakAgreement request allows the simulator to install inconsistent per-party shares, but only if it provides, for each party, an externally verifiable proof accusing a set of at least d parties (line 33). In this case, the functionality emits corresponding detection events, explicitly exposing the disagreement and attributing blame.

Role of metadata (now included). We now explain the role of the metadata components previously ignored. At sharing time, the functionality samples metadata randomness (r_D is actually chosen by a corrupted dealer) and computes a metadata value m that is distributed to all parties and to the simulator (line 16). This metadata serves as a sharing-time handle that can be externally related to the shared secrets.

In the event of a BreakAgreement, the functionality recomputes per-party metadata values from the substituted shares and explicitly provided randomness, allowing metadata views to diverge only in conjunction with a justified accountability event (line 33).

The primary motivation for metadata is to enable *co-sharing*: multiple instances of the functionality (or different sharing schemes) can be linked at sharing time by proving relations between their metadata. This is particularly useful for mask opening and share migration, where parties jointly open masked values in one sharing scheme and re-encode them in another, while ensuring that both sharings correspond to the same underlying secret.

4.2. UC Shamir-type Accountable PBHACSS

In this section, we describe our Shamir-type realization $\mathcal{P}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ (see Algorithm 1) of the $\mathcal{F}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}(\mathsf{Mt}^{\mathsf{AH}})$ functionality, presented in Functionality 1, where $\mathsf{Mt}^{\mathsf{AH}} : (\mathbf{s}, \mathbf{r}) \in \mathbb{F}_p^K \mapsto \boldsymbol{c}_0$ s.t. $\forall k \in [K], \boldsymbol{c}_0[k] = \mathsf{TC.Com}^{\mathsf{srs}_0}(s, r)$ (The formal definition of the trapdoor commitment scheme TC is given in Appendix G).

TPC sharing interface. For a trapdoor polynomial commitment scheme TPC with structured reference string srs and a vector of pairwise distinct evaluation points $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$, we define the sharing operator

$$(\boldsymbol{\phi},\, c,\, \boldsymbol{\theta}) \;\leftarrow\; \mathsf{TPC.Share}^{\mathsf{srs}}_{\boldsymbol{\alpha}}(\boldsymbol{\xi}),$$

where $\boldsymbol{\xi} \in \mathbb{F}^{\ell+1}$ is interpreted as a polynomial of degree at most ℓ. The output consists of:

- $c := \mathsf{TPC.Com}^{\mathsf{srs}}(\boldsymbol{\xi}) \in \mathbb{H}_C$,
- $\boldsymbol{\phi} := \mathsf{eval}_{\boldsymbol{\alpha}}(\boldsymbol{\xi}) \in \mathbb{F}^n$,
- $\boldsymbol{\theta}[i] := \mathsf{TPC.Open}^{\mathsf{srs}}_{\alpha_i}(c,\, \boldsymbol{\phi}[i]) \in \mathbb{H}_\theta$ for each $i \in [n]$.

When public keys $\mathbf{pk} = (\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and an encryption randomness vector $\boldsymbol{r}_e \in (\mathsf{Rand}_{\mathsf{Enc}})^{2n}$ are available, we extend the interface to:

$$(\hat{\boldsymbol{\phi}},\, c,\, \hat{\boldsymbol{\theta}}) \;\leftarrow\; \mathsf{TPC.EncShare}^{\mathsf{srs}}_{\boldsymbol{\alpha},\mathbf{pk}}(\boldsymbol{\xi}, \boldsymbol{r}_e),$$

which produces the same commitment c, together with element-wise encryptions, i.e. $\forall i \in [n]$:

$$\hat{\boldsymbol{\phi}}[i] := \mathsf{Enc}_{\mathsf{pk}_i}\big(\boldsymbol{\phi}[i];\, \boldsymbol{r}_e[i]\big), \quad \hat{\boldsymbol{\theta}}[i] := \mathsf{Enc}_{\mathsf{pk}_i}\big(\boldsymbol{\theta}[i];\, \boldsymbol{r}_e[i{+}n]\big).$$

Algorithm 1 $\mathcal{P}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ – Sharing phase – First Part

Parameters: $n, t, K \in \mathbb{N}$; reconstruction threshold $\ell \in [t,\, n{-}t)$.

Use: $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$, $\mathcal{F}_{\mathrm{zkpok}}(\mathcal{R}^{(\boldsymbol{\alpha},\ell)}_{\mathrm{vepc}})$, $\mathcal{F}_{\mathrm{ack}}$.

Dealer D with inputs $\mathbf{s} = [s_1, \ldots, s_K] \in \mathbb{F}^K$

Sample encryption randomness $\mathbf{R}^*_e \in \mathsf{Mat}_{(n,4n)}(\mathsf{Rand}_{\mathsf{CT}})$.

Sample coefficient matrices
$(\boldsymbol{\Xi}, \boldsymbol{\Xi}_{r_0}) \in \mathsf{Mat}_{(\ell+1,K)}(\mathbb{F}) \times \mathsf{Mat}_{(\ell+1,K)}(\mathbb{F})$ such that:
$\boldsymbol{\Xi}[1,k] \leftarrow s_k$ and $\boldsymbol{\Xi}_{r_0}[1,k] \leftarrow_{\$} \mathbb{F}$ for all $k \in [K]$,
$\boldsymbol{\Xi}[d,k], \boldsymbol{\Xi}_{r_0}[d,k] \leftarrow_{\$} \mathbb{F}$ for all
$(d,k) \in \{2, \ldots, \ell+1\} \times [K]$.

Apply the Vandermonde-based expansion row-wise:
$(\boldsymbol{\Xi}^*, \boldsymbol{\Xi}^*_{r_0}) \leftarrow \big(\mathsf{Encode}_{\boldsymbol{\alpha}}(\boldsymbol{\Xi}),$
$\mathsf{Encode}_{\boldsymbol{\alpha}}(\boldsymbol{\Xi}_{r_0})\big)$.

Per-recipient VEPC proofs.

For all $j \in [n]$:

$wit^j_D := \big(\mathbf{R}^*_e[:,j], \boldsymbol{\Xi}^*[:,j], \boldsymbol{\Xi}^*_{r_0}[:,j]\big)$,

$stmt^j := \big(\mathbf{c}^*_0[j], \mathbf{c}^*[j], \hat{\boldsymbol{\Theta}}^*[:,j], \hat{\boldsymbol{\Phi}}^*[:,j],$
$\mathbf{c}^*_{r_0}[j], \hat{\boldsymbol{\Theta}}^*_{r_0}[:,j], \hat{\boldsymbol{\Phi}}^*_{r_0}[:,j]\big)$
$:= \psi^{\boldsymbol{\alpha},\mathbf{pk}}_{vepc}(wit^j_D)$.

invoke $\mathcal{F}_{\mathrm{zkpok}}(\mathcal{R}^{(\boldsymbol{\alpha},\ell)}_{\mathrm{vepc}}).(\mathsf{Prove}, \mathsf{sid}_{(D,j)},$
$(wit^j_D, stmt^j), p_j)$.

Accountable broadcast of commitments.

Invoke $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}.(\mathsf{rb_bcast}, \mathsf{sid}_D, \mathbf{c}^*_0, \mathbf{c}^*, \mathbf{c}^*_{r_0})$.

Process p_j: handling the dealer's sharing

Upon $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}.(\mathsf{deliver}, \mathsf{sid}_D, \mathbf{c}^*_0, \mathbf{c}^*, \mathbf{c}^*_{r_0})$,
store $(\mathbf{c}^*_0, \mathbf{c}^*, \mathbf{c}^*_{r_0})$.

Upon successful verification of
$\mathcal{F}_{\mathrm{zkpok}}(\mathcal{R}^{(\boldsymbol{\alpha},\ell)}_{\mathrm{vepc}}).(\mathsf{Verify}, \mathsf{sid}_{(D,j)}, stmt^j, p_j)$
and consistency of the commitments with $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$:
$\mathcal{F}_{\mathrm{ack}}.(\mathsf{approve}, \mathsf{sid}_D)$
and forward to each p_i the chunk
$\big(\hat{\boldsymbol{\Phi}}^*[i,j],$
$\hat{\boldsymbol{\Theta}}^*[i,j],$
$\hat{\boldsymbol{\Phi}}^*_{r_0}[i,j],$
$\hat{\boldsymbol{\Theta}}^*_{r_0}[i,j], i, j, \mathsf{sid}_D\big)$.

In this case, $(\boldsymbol{\phi}, c, \boldsymbol{\theta}) = \mathsf{TPC.Share}^{\mathsf{srs}}_{\boldsymbol{\alpha}}(\boldsymbol{\xi})$ is implicitly defined.

VEPC relation. Let $\mathbb{F}$ be a field of prime order, $\ell \geq 0$, $K, n \in \mathbb{N}$, and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$ be distinct. Assume that both Enc, TPC, and TC are $\mathbb{F}_p$-additively homomorphic in each input parameter (plaintext, randomness, polynomial).

message and randomness. Define $\psi^{\boldsymbol{\alpha},\mathbf{pk}}_{vepc} : (\mathbf{r}_e,\ \boldsymbol{\xi}, \boldsymbol{\xi}_{r_0}) \mapsto (c_0, c, \hat{\boldsymbol{\theta}},\ \hat{\boldsymbol{\phi}},\ c_{r_0},\ \hat{\boldsymbol{\theta}}_{r_0},\ \hat{\boldsymbol{\phi}}_{r_0})$ such that:

$$(\hat{\boldsymbol{\phi}}, c, \hat{\boldsymbol{\theta}}) \;=\; \mathsf{TPC.EncShare}^{\mathsf{srs}}_{\boldsymbol{\alpha},\mathbf{pk}}\big(\boldsymbol{\xi}\ ;\ \boldsymbol{r}_e[1:2n]\big) \qquad \text{(Secret-Sharing)}$$

$$(\hat{\boldsymbol{\phi}}_{r_0}, c_{r_0}, \hat{\boldsymbol{\theta}}_{r_0}) \;=\; \mathsf{TPC.EncShare}^{\mathsf{srs}}_{\boldsymbol{\alpha},\mathbf{pk}}\big(\boldsymbol{\xi}_{r_0}\ ;\ \boldsymbol{r}_e[2n+1:4n]\big) \qquad \text{(Rand-Sharing)}$$

$$c_0 \;=\; \mathsf{TC.Com}^{\mathsf{srs}_0}\big(\boldsymbol{\xi}[1]\ ;\ \boldsymbol{\xi}_{r_0}[1]\big) \;\in \mathbb{H}^0_C, \qquad \text{(0-Commitment)}$$

Consequently:

$$\forall i \in [n] : \mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_i}\big(c,\, \boldsymbol{\phi}[i],\, \boldsymbol{\theta}[i]\big) \;=\; \mathsf{true}. \quad \text{(Verify)}$$

$$\forall i \in [n] : \mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_i}\big(c_{r_0},\, \boldsymbol{\phi}_{r_0}[i],\, \boldsymbol{\theta}_{r_0}[i]\big) \;=\; \mathsf{true}. \qquad (r_0\text{-Verify})$$

Algorithm 2 $\mathcal{P}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ – Sharing phase – Second Part

Forwarding & local decoding at p_i
At p_i, maintain a set $J_i \subseteq [n]$ of indices j with valid chunks.
Upon receipt of
  $(\hat{\mathbf{\Phi}}^*[i,j], \hat{\mathbf{\Theta}}^*[i,j], \hat{\mathbf{\Phi}}^*_{r_0}[i,j]$, from p_j:
  $\hat{\mathbf{\Theta}}^*_{r_0}[i,j], i, j, \mathsf{sid}_D)$
  For each $\mathbf{X} \in \{\mathbf{\Phi}, \mathbf{\Theta}, \mathbf{\Phi}_{r_0}, \mathbf{\Theta}_{r_0}\}$, set
    $\mathbf{X}^*[i,j] \leftarrow \mathsf{Dec}_{\mathsf{sk}_i}(\hat{\mathbf{X}}^*[i,j])$.
  If
    $\mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_j}(\mathbf{c}^*[j], \mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j])$
    $\wedge\ \mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_i}(\mathbf{c}^*_{r_0}[j], \mathbf{\Phi}^*_{r_0}[i,j], \mathbf{\Theta}^*_{r_0}[i,j])$
    then add j to J_i.

Upon $|J_i| = K$:
  Recover row i via decoding:
    $(\mathbf{\Phi}[i,:], \mathbf{\Theta}[i,:]) \leftarrow \mathsf{Decode}_{(\boldsymbol{\alpha}, J_i)}(\mathbf{\Phi}^*_{J_i}[i,:], \mathbf{\Theta}^*_{J_i}[i,:])$,
    $(\mathbf{\Phi}_{r_0}[i,:], \mathbf{\Theta}_{r_0}[i,:]) \leftarrow \mathsf{Decode}_{(\boldsymbol{\alpha}, J_i)}(\mathbf{\Phi}^*_{r_0,J_i}[i,:], \mathbf{\Theta}^*_{r_0,J_i}[i,:])$.
  (If needed) similarly decode
    $\mathbf{c}_0, \mathbf{c}, \mathbf{c}_{r_0}$ from $(\mathbf{c}^*_0, \mathbf{c}^*, \mathbf{c}^*_{r_0})$.
  For all $k \in [K]$, set
    $share[\mathsf{sid}_{(D,k)}] \leftarrow (\mathbf{c}_0[k], \mathbf{c}[k], \mathbf{\Phi}[i,k], \mathbf{\Theta}[i,k], \mathbf{c}_{r_0}[k], \mathbf{\Phi}_{r_0}[i,k], \mathbf{\Theta}_{r_0}[i,k])$.

Reception notification (ack gating).
When $(\mathbf{\Phi}[i,:], \mathbf{\Theta}[i,:]) \neq (\perp, \perp)$ and $\mathcal{F}_{\mathrm{ack}}.(\mathsf{acknowledged}, \mathsf{sid}_D)$:
  trigger $(\mathsf{Sh.receive}, \mathsf{sid}_D, (\mathbf{c}_0, \mathbf{c}, \mathbf{c}_{r_0}))$.

Accountability
upon notification $(\mathsf{detection}, \mathsf{sid}_D, G, \pi)$ **from** $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$:
  trigger notification $(\mathsf{detection}, \mathsf{sid}_D, G, \pi)$.

Local additively homomorphic operation
upon request $(\mathsf{local_add}, \mathsf{ssid}_{sum}, [\mathsf{ssid}_1, \ldots, \mathsf{ssid}_S], \mathbf{x} \in \mathbb{F}^S)$:
  $share[\mathsf{ssid}_{sum}] \leftarrow \bigoplus_{u \in [S]}(\mathbf{x}[u] \odot share[\mathsf{ssid}_u])$.

Define $\mathcal{R}^{(\boldsymbol{\alpha},\ell)}_{\mathrm{vepc}}$ as the set of pairs of tuples of the form:

$$\Big((\mathbf{r}_e,\ \boldsymbol{\xi}, \boldsymbol{\xi}_{r_0}),\ \psi^{\boldsymbol{\alpha},\mathbf{pk}}_{vepc}(\mathbf{r}_e,\ \boldsymbol{\xi}, \boldsymbol{\xi}_{r_0})\Big)$$

Here $\boldsymbol{\xi} \in \mathbb{F}^{\ell+1}$ (resp. $\boldsymbol{\xi}_{r_0} \in \mathbb{F}^{\ell+1}$) is the coefficient vector of the polynomial sharing the secret $\boldsymbol{\xi}[1]$ (resp. the randomness $\boldsymbol{\xi}_{r_0}[1]$), which is committed in c (resp. c_{r_0}). The vectors $\boldsymbol{\phi}, \boldsymbol{\phi}_{r_0} \in \mathbb{F}^n$ are their evaluations at the points $\boldsymbol{\alpha}$, and $\boldsymbol{\theta}, \boldsymbol{\theta}_{r_0}$ are the corresponding opening proofs. For any $\boldsymbol{x} \in \{\boldsymbol{\phi}, \boldsymbol{\phi}_{r_0}, \boldsymbol{\theta}, \boldsymbol{\theta}_{r_0}\}$, we denote by $\hat{\boldsymbol{x}}$ its element-wise encryption under the respective recipients' public keys, using randomness taken from $\mathbf{r}_e$. We omit the (random-free) polynomial-commitment randomness for brevity. The commitment $c_0 = \mathsf{TPC.Com}^{\mathsf{srs}_0}(\boldsymbol{\xi}[1]\ ;\ \boldsymbol{\xi}_{r_0}[1])$ will be crucial for efficient (and still robust in the degraded case) reconstruction. Particular care is needed for UC security.

Dealer computation (batched secret sharing, mirrors encoding). The dealer samples $2K$ polynomials of degree $\leq \ell$ and packs their coefficients as $(\mathbf{\Xi}, \mathbf{\Xi}_{r_0}) \in \mathsf{Mat}_{(\ell+1,K)}(\mathbb{F})^2$, with $\mathbf{\Xi}[1,k]$ the k-th secret, and $\mathbf{\Xi}_{r_0}[1,k]$ the k-th randomness used in the k-th secret's commitment. It samples $\mathbf{R}^*_e \in \mathsf{Mat}_{(4n,n)}(\mathsf{Rand}_{\mathsf{Enc}})$ too. Encode rows by:

$$(\mathbf{\Xi}^*, \mathbf{\Xi}^*_{r_0}) = \big(\mathsf{Encode}_{\boldsymbol{\alpha}}(\mathbf{\Xi}), \mathsf{Encode}_{\boldsymbol{\alpha}}(\mathbf{\Xi}_{r_0})\big)$$

Here $\mathsf{Encode}_{\boldsymbol{\alpha}}$ denotes the linear $[n,K]$ Vandermonde code applied row-wise, i.e., $\mathsf{Encode}_{\boldsymbol{\alpha}} : \mathsf{Mat}_{(m,K)}(\mathbb{F}) \to \mathsf{Mat}_{(m,n)}(\mathbb{F})$ for any m.

Algorithm 3 $\mathcal{P}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}$ – Reconstruction phase

Reconstruction interface
  (dual-threshold)
upon request $(\mathsf{Rec.enable}, \mathbf{sid})$ at p_i, with $\mathbf{sid} \in \mathsf{SID}^{K'}$:
  For all $k \in [K']$:
    $(\mathbf{c}_0[k], \mathbf{c}[k], \mathbf{\Phi}[i,k], \mathbf{\Theta}[i,k], \mathbf{c}_{r_0}[k], \mathbf{\Phi}_{r_0}[i,k], \mathbf{\Theta}_{r_0}[i,k]) \leftarrow share[\mathbf{sid}[k]]$.
  Re-encode local row i via Vandermonde expansion:
    $(\mathbf{c}^*_0, \mathbf{c}^*, \mathbf{c}^*_{r_0}, \mathbf{\Phi}^*[i,:], \mathbf{\Theta}^*[i,:], \mathbf{\Phi}^*_{r_0}[i,:], \mathbf{\Theta}^*_{r_0}[i,:]) \leftarrow \mathsf{Encode}_{\boldsymbol{\alpha}}(\mathbf{c}_0, \mathbf{c}, \mathbf{c}_{r_0}, \mathbf{\Phi}[i,:], \mathbf{\Theta}[i,:], \mathbf{\Phi}_{r_0}[i,:], \mathbf{\Theta}_{r_0}[i,:])$.
  For all $j \in [n]$, send to p_j:
    $(\mathbf{sid}^{(i,j)}, \mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j], \mathbf{\Phi}^*_{r_0}[i,j], \mathbf{\Theta}^*_{r_0}[i,j])$.

Row decoding at p_j and encoded secret broadcasting
At p_j, maintain a sender set $Q_j(\mathbf{sid}) \subseteq [n]$ for which valid chunks were received.
upon receipt $(\mathbf{sid}^{(i,j)}, \mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j]$, from p_i:
  $\mathbf{\Phi}^*_{r_0}[i,j], \mathbf{\Theta}^*_{r_0}[i,j])$
  If
    $\mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_j}(\mathbf{c}^*[i], \mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j])$
    $\wedge\ \mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_j}(\mathbf{c}^*_{r_0}[i], \mathbf{\Phi}^*_{r_0}[i,j], \mathbf{\Theta}^*_{r_0}[i,j])$
  add i to $Q_j(\mathbf{sid})$ and store
    $(\mathbf{\Phi}^*[i,j], \mathbf{\Theta}^*[i,j], \mathbf{\Phi}^*_{r_0}[i,j], \mathbf{\Theta}^*_{r_0}[i,j])$.
when $|Q_j(\mathbf{sid})| \geq K'$:
  Interpolate the encoded coefficient for this column j:
    $\mathbf{\Xi}^*[1,j] \leftarrow \mathsf{Interpol}(\mathbf{\Phi}^*[Q_j(\mathbf{sid}), j])$,
    $\mathbf{\Xi}^*_{r_0}[1,j] \leftarrow \mathsf{Interpol}(\mathbf{\Phi}^*_{r_0}[Q_j(\mathbf{sid}), j])$.
  Broadcast $(\mathbf{sid}^j, \mathbf{\Xi}^*[1,j], \mathbf{\Xi}^*_{r_0}[1,j])$ to all parties.

Erasure decoding of the batch at p_i
At p_i, maintain a sender set $Q'_i(\mathbf{sid}) \subseteq [n]$ for which valid encoded secrets were received.
upon receipt $(\mathbf{sid}^j, \mathbf{\Xi}^*[1,j], \mathbf{\Xi}^*_{r_0}[1,j])$ from p_j:
  If
    $\mathsf{TC.Verify}^{\mathsf{srs}_0}(\mathbf{c}^*_0[j], \mathbf{\Xi}^*[1,j], \mathbf{\Xi}^*_{r_0}[1,j]) = true$:
  add j to $Q'_i(\mathbf{sid})$ and store
    $(\mathbf{\Xi}^*[1,j], \mathbf{\Xi}^*_{r_0}[1,j])$.
when $|Q'_i(\mathbf{sid})| = \ell$:
  Recover the first coefficient row:
    $\mathbf{\Xi}[1,:] \leftarrow \mathsf{Decode}_{(\boldsymbol{\alpha}, Q'_i(\mathbf{sid}))}(\mathbf{\Xi}^*[1, Q'_i(\mathbf{sid})])$.
  For all $k \in [K']$:
    trigger $(\mathsf{Rec.retrieve}, \mathbf{sid}[k], \mathbf{\Xi}[1,k])$.

Then, it computes $(\mathbf{c}^*_0, \mathbf{c}^*, \hat{\mathbf{\Theta}}^*, \hat{\mathbf{\Phi}}^*, \mathbf{c}^*_{r_0}, \hat{\mathbf{\Theta}}^*_{r_0}, \hat{\mathbf{\Phi}}^*_{r_0})$, such that, for each $j \in [n]$: $stmt_j = \psi^{\boldsymbol{\alpha},\mathbf{pk}}_{vepc}(wit_j) = (\mathbf{c}^*_0[j], \mathbf{c}^*[j], \hat{\mathbf{\Theta}}^*[:,j], \hat{\mathbf{\Phi}}^*[:,j], \mathbf{c}^*_{r_0}[j], \hat{\mathbf{\Theta}}^*_{r_0}[:,j], \hat{\mathbf{\Phi}}^*_{r_0}[:,j])$,
with $wit_j = (\mathbf{R}^*_e[:,j],\ \mathbf{\Xi}^*[:,j], \mathbf{\Xi}^*_{r_0}[:,j])$

The dealer reliably broadcasts $(\mathbf{c}^*_0, \mathbf{c}^*, \mathbf{c}^*_{r_0})$ via $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$ and, $\forall j$, proves to p_j via $\mathcal{F}_{\mathrm{zkpok}}$ that $(wit_j, stmt_j) \in \mathcal{R}^{(\boldsymbol{\alpha},\ell)}_{\mathrm{vepc}}$

Receiver checks and forwarding. If p_j is convinced (and the commitments used in the proof match $(\mathbf{c}^*_0[j], \mathbf{c}^*[j], \mathbf{c}^*_{r_0}[j])$ delivered by $\mathcal{F}^{\mathrm{acc}}_{\mathrm{rbc}}$), it sends to each p_i the chunk $\big(\hat{\mathbf{\Theta}}^*[i,j], \hat{\mathbf{\Phi}}^*[i,j], \hat{\mathbf{\Theta}}^*_{r_0}[i,j], \hat{\mathbf{\Phi}}^*_{r_0}[i,j]\big)$, and then *approves* $\mathcal{F}_{\mathrm{ack}}$ (see Functionality 9), which can be easily realized from $\mathcal{P}_{\mathrm{ack}}$ (see Algorithm 10). Upon receiving $\big(\hat{\mathbf{\Theta}}^*[i,j], \hat{\mathbf{\Phi}}^*[i,j], \hat{\mathbf{\Theta}}^*_{r_0}[i,j], \hat{\mathbf{\Phi}}^*_{r_0}[i,j]\big)$ from p_j, party p_i computes $\mathbf{X}^*[i,j] = \mathsf{Dec}_{\mathsf{sk}_i}(\hat{\mathbf{X}}^*[i,j])$ for each $\mathbf{X} \in \{\mathbf{\Phi}, \mathbf{\Theta}, \mathbf{\Phi}_{r_0}, \mathbf{\Theta}_{r_0}\}$.
and checks both:

$$\mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_i}\big(\mathbf{c}^*[j],\ \mathbf{\Phi}^*[i,j],\ \mathbf{\Theta}^*[i,j]\big)$$

$$\mathsf{TPC.Verify}^{\mathsf{srs}}_{\alpha_i}\big(\mathbf{c}^*_{r_0}[j],\ \mathbf{\Phi}^*_{r_0}[i,j],\ \mathbf{\Theta}^*_{r_0}[i,j]\big).$$

Decoding from K verified rows. Let $J \subseteq [n]$, $|J| = K$, be indices of verified columns for row i. Recover the base objects by the same coefficients:

$$\mathbf{\Phi}[i,:] \;=\; \mathsf{Decode}_{(\boldsymbol{\alpha},J)}(\mathbf{\Phi}^*_J[i,:])$$
$$\mathbf{\Theta}[i,:] \;=\; \mathsf{Decode}_{(\boldsymbol{\alpha},J)}(\mathbf{\Theta}^*_J[i,:]).$$

If $(\mathbf{c}_0, \mathbf{c}, \mathbf{c}_{r_0})$ was not separately RBC'ed, one can analogously recover, $\tilde{\mathbf{c}} = \mathsf{decode}_{(\boldsymbol{\alpha},J)}(\tilde{\mathbf{c}}^*_J)$, $\forall \tilde{\mathbf{c}} \in \{\mathbf{c}_0, \mathbf{c}, \mathbf{c}_{r_0}\}$,

To ensure completeness, process p_i triggers the reception notification only upon the $\mathcal{F}_{\text{ack}}$'s acknowledgement notification. Hence,
$(\mathbf{c}_0[k],\ \mathbf{c}[k],\ \mathbf{\Phi}[i,k],\ \mathbf{\Theta}[i,k],\ \mathbf{c}_{r_0}[k], \mathbf{\Phi}_{r_0}[i,k],\ \mathbf{\Theta}_{r_0}[i,k])$ is obtained by process p_i for each $k \in [K]$, to be reused during reconstruction via erasure codes. This avoids online error-correction despite the high polynomial degree.

Reconstruction. Suppose the parties aim to reconstruct a batch of K secrets shared through material $(\mathbf{c}_0,\ \mathbf{c},\ \mathbf{\Phi},\ \mathbf{\Theta},\ \mathbf{c}_{r_0}, \mathbf{\Phi}_{r_0},\ \mathbf{\Theta}_{r_0})$, as defined above, where each process p_i locally holds $(\mathbf{c}_0,\ \mathbf{c},\ \mathbf{\Phi}[i,:],\ \mathbf{\Theta}[i,:],\ \mathbf{c}_{r_0}, \mathbf{\Phi}_{r_0}[i,:],\ \mathbf{\Theta}_{r_0}[i,:])$. We emphasize that the batch need not originate from a single sharing instance: any $\mathbb{F}_p$-linear combination of sharings yields a valid batch thanks to the $\mathbb{F}_p$-linearity of the construction, inherited from the $\mathbb{F}_p$-linearity of the underlying polynomial commitments.

Reconstruction mirrors the sharing phase in a fully leaderless manner. Each process p_i encodes its local row into $(\mathbf{c}^*_0,\ \mathbf{c}^*,\ \mathbf{\Phi}^*[i,:],\ \mathbf{\Theta}^*[i,:],\ \mathbf{c}^*_{r_0}, \mathbf{\Phi}^*_{r_0}[i,:],\ \mathbf{\Theta}^*_{r_0}[i,:])$ by reapplying the Vandermonde-based expansion. It then forwards to each p_j the tuple $(\mathbf{\Phi}[i,j]^*,\ \mathbf{\Theta}[i,j]^*,\ \mathbf{\Phi}_{r_0}[i,j]^*,\ \mathbf{\Theta}_{r_0}[i,j]^*)$, who verifies the two valuation chunks against $(\mathbf{c}^*[j], \mathbf{c}^*_{r_0}[j])$ using the corresponding proof $(\mathbf{\Theta}^*[i,j], \mathbf{\Theta}^*_{r_0}[i,j])$.

Upon collecting $n - t$ valid chunks, process p_j interpolates the pair of corresponding columns $(\mathbf{\Xi}^*[:,j],\ \mathbf{\Xi}^*_{r_0}[:,j])$ and broadcasts the candidate secret chunk $\mathbf{\Xi}[1,j]^*$ accompanied by the randomness $\mathbf{\Xi}^*_{r_0}[1,j]$, whose correctness can be checked by everyone against $\mathbf{c}^*_0[j]$ via $\mathsf{TC}^{\mathsf{srs}_0}.\mathsf{Verify}(\mathbf{c}^*_0[j], \mathbf{\Xi}^*[1,j], \mathbf{\Xi}^*_{r_0}[1,j])$. Finally, once K valid codewords are gathered, each party applies the erasure-decoding step to recover the coefficient matrix $\mathbf{\Xi}$, and thereby retrieve the K original secrets $\{\mathbf{\Xi}[1,k]\}_{k=1}^{K}$.

Theorem 4. *(Theorem 1) Let $t_{\text{acc}} < \ell \in [t, n-t)$. $\mathcal{P}^{\text{AH,acc}}_{\text{pbhacss}}$ (Algorithm 1) UC-realizes $\mathcal{F}^{\text{AH,acc}}_{\text{pbhacss}}(\mathsf{Mt}^{\mathsf{AH}})$ (Functionality 1) in the $(\mathcal{F}^{\text{acc}}_{\text{rbc}}, \mathcal{F}_{\text{zkpok}}(\mathcal{R}^{(\boldsymbol{\alpha},\ell)}_{\text{vepc}}), \mathcal{F}_{\text{ack}})$-hybrid model with $O(1)$ rounds and $O(Kn\kappa)$ communication for batch of $K \in \Omega(n)$ sharings.*

The proof is given in Appendix C.

5. Online Phase

Functionality overview. The functionality $\mathcal{F}^{\text{acc}}_{\text{ampc}}$ (Functionality 2) evaluates circuit $\mathcal{C}$, with default inputs for parties that do not provide an input (line 1– 2). A partition map μ assigns each process to a partition, and is initialized to the trivial single-partition setting $\mu(i) = 1$ for all parties (line 4).

Functionality 2 $\mathcal{F}^{\text{acc}}_{\text{ampc}}$

1: **Parameters:**
2: $n, t, t_{acc} \in \mathbb{N}$; arithmetic circuit $\mathcal{C}$ over $\mathbb{F}$; default inputs $\boldsymbol{I}^{\mathsf{def}} = (\boldsymbol{\iota}^{\mathsf{def}}_1, \ldots, \boldsymbol{\iota}^{\mathsf{def}}_n)$

3: **Variables:**
4: $\mathsf{Map}(\mathsf{PID} \to [\mathsf{n}])\ \mu$ (init $\forall i \in [n], \mu(i) = 1$) ▷ Partitioning when $f > t$
5: $\mathsf{Map}([n] \times \mathsf{PID} \to \mathbb{F}^{k_i})\ \mathit{Inputs}[][]$ (init $\perp$)
6: $\mathsf{Map}([n] \to \mathsf{Set}(\mathsf{PID}))\ \mathit{Provided}[]$ (init $\emptyset$)
7: $\mathsf{Map}([n] \to \mathsf{Bool})\ \mathit{Computed}[]$ (init **false**)
8: $\mathsf{Bool}\ \mathit{broken}$ (init **false**)

9: **Interface:**
10: *request* $(\mathsf{Input}, k \in [n], \boldsymbol{\iota}_i)$ from party p_i ▷ $k \neq 1$ anticipates equivocations
11: *notification* $(\mathsf{Output}, \boldsymbol{o}_i)$ to party p_i
12: *request* $(\mathsf{ChooseSet}, k \in [n], Q \subseteq [n], |Q| \geq n - t)$ from Sim
13: *request* $(\mathsf{BreakLiveness})$ from Sim
14: *request* $(\mathsf{BreakAgreement}, \mu^*, \boldsymbol{\pi})$ from Sim
15: *notification* $(\mathsf{detect}, G, \pi^{pom})$

16: **Functionality:**
17: **Upon receiving** $(\mathsf{Input}, k, \boldsymbol{\iota}_i)$ from p_i for the first time for this (i, k):
18: **if** $k \neq 1$ **and** $\mathit{broken} = \mathit{false}$, **then:** $K \leftarrow \emptyset$ ▷ partitioning requires detection
19: **else if** $p_i \in$ Corrupted, **then:** $K \leftarrow \{k\}$ ▷ corrupted processes can equivocate
20: **else:** $K \leftarrow [n]$ ▷ same honest input in all partitions
21: $\forall k' \in K$: $\mathit{Inputs}[k'][i] \leftarrow \boldsymbol{\iota}_i$; $\mathit{Provided}[k'] \leftarrow \mathit{Provided}[k'] \cup \{p_i\}$
22: **Notify** Sim of (Input, p_i)

23: **Upon receiving** $(\mathsf{ChooseSet}, k, Q)$ from Sim with $|Q| \geq n - t$, $Q \subseteq \mathit{Provided}[k]$:
24: **If** $\mathit{Computed}[k] = \mathit{false}$, **and** $(k = 1$ **or** $\mathit{broken} = \mathit{true})$:
25: **For all** $i \in [n]$:
26: $\boldsymbol{x}^k_i \leftarrow \begin{cases} \mathit{Inputs}[k][i] & \text{if } i \in Q \\ \boldsymbol{\iota}^{\mathsf{def}}_i & \text{otherwise} \end{cases}$
27: Compute $(\boldsymbol{o}^k_1, \ldots, \boldsymbol{o}^k_n) \leftarrow \mathcal{C}(\boldsymbol{x}^k_1, \ldots, \boldsymbol{x}^k_n)$ ▷ Correct partitionwise
28: $\mathit{Computed}[k] \leftarrow$ **true**
29: **For each** $i \in [n]$, **if** $\mu(i) = k$ **or** $p_i \in$ Corrupted:
30: **Send** $(\mathsf{Output}, \boldsymbol{o}^k_i)$ to p_i as a delayed output.

31: **Upon receiving** $(\mathsf{BreakLiveness})$ from Sim with $f > t$: ▷ Liveness Violation
32: Any notification (except detection) may be indefinitely delayed.

33: **Upon receiving** $(\mathsf{BreakAgreement}, \mu^* : [n] \to [n], \boldsymbol{\pi})$ from Sim with $f > t$:

34: **If** $\forall i \in [n]$, the judge returns $\mathsf{dis}(G_i)$ on top of $\boldsymbol{\pi}[i]$ with $|G_i| \geq n - 2t$:

35: $\mu \leftarrow \mu^*$; $\mathit{broken} \leftarrow \mathit{true}$ ▷ process p_i belongs to $\mu(i)$-th partition
36: **If** $f \leq t_{acc}$, **then: for every** $p_i \in \Psi$:
37: **Send** $(\mathsf{detect}, G_i, \boldsymbol{\pi}[i])$ to p_i as a delayed output. ▷ Accountability

Inputs. Parties submit inputs through the interface $(\mathsf{Input}, k, \boldsymbol{\iota}_i)$ (line 10). Before an agreement break is declared, only $k = 1$ is accepted (line 18). After a break, corrupted parties may equivocate across partitions by choosing distinct inputs per k, whereas honest parties are interpreted as having the same input in all partitions (line 19– 20).

Qualified set and outputs. The simulator selects a qualified set Q of $n - t$ input providers for a given partition k via $(\mathsf{ChooseSet}, k, Q)$ (line 12). Correctness is defined partitionwise: the functionality computes outputs $(\boldsymbol{o}^k_1, \ldots, \boldsymbol{o}^k_n)$ for each partition k independently (line 27). Outputs are then released (as delayed outputs) to all corrupted parties, and to correct parties whose partition assignment matches k under μ (line 29– 30).

Break conditions and accountability. In case of a valid

Algorithm 4 $\mathcal{P}^{\text{acc}}_{\text{ampc}}$ – Online realization of functionality $\mathcal{F}^{\text{acc}}_{\text{ampc}}$

Parameters: $n, t, t_{acc} \in \mathbb{N}$; finite field $(\mathbb{F}, \oplus_{\mathbb{F}}, \otimes_{\mathbb{F}}, 0_{\mathbb{F}}, 1_{\mathbb{F}})$; $B = c_M + \lceil c_R/2 \rceil$ arithmetic circuit $\mathcal{C} = (\mathcal{G}, \mathcal{W}, \mathcal{L})$ with multiplicative layers $L_0, \ldots, L_D$
Use: Ideal functionalities $\mathcal{F}^{\text{acc}}_{\text{idx_acs}}$, $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$, $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle$

Offline Phase
1: Invoke $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle$.(Gen.enable, sid).
2: Wait until being notified $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle$.(Gen.receive, sid). ▷ Obtain B Beaver triples indexed by $k \in [B]$, with SSIDs $(\mathsf{sid}_{(a,k)}, \mathsf{sid}_{(b,k)}, \mathsf{sid}_{(c,k)})$ assigned to gate g with $\mathsf{midx}(g) = k \le c_M$. The remaining indices are used to supply the c_R random input gates.

Input sharing ▷ The case with inputs as tuples over $\mathbb{F}$ can be adapted trivially
3: **Upon request** (Input, sid, x_i):
4: Invoke $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$.(Sh.share, sid_i, x_i)
5: Upon notification $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$.(Sh.receive, sid_j):
6: send (acs_validate, sid, p_j) to $\mathcal{F}^{\text{acc}}_{\text{idx_acs}}$.
7: Upon $\mathcal{F}^{\text{acc}}_{\text{idx_acs}}$.(acs_decide, sid, Q): ▷ $|Q| = n - t$
8: **trigger** Add_Eval(0)

Layer-by-layer evaluation
9: **For** $\ell \in [1 : D-1]$:
10: **trigger** Mult_Eval(ℓ) ▷ Requires communication to open the Beaver triples
11: **trigger** Add_Eval(ℓ) ▷ Local computation only

Output Reconstruction
12: Help each process to reconstruct its associated output wires
13: **Trigger notification** (Output, sid, o_i) accordingly

Additive Evaluation
14: Upon Add_Eval($\ell \in [1 : D]$):
15: Let $\mathcal{G}^A_\ell = \mathcal{G}^A \cap \mathcal{G}_\ell$, $k^{\min}_{(a,\ell)} = \min_{g \in \mathcal{G}^A_\ell} \mathsf{idx_a}(g)$, $k^{\max}_{(a,\ell)} = \max_{g \in \mathcal{G}^A_\ell} \mathsf{idx_a}(g)$
16: $\forall k \in [k^{\min}_{(a,\ell)} : k^{\max}_{(a,\ell)}]$, for $g \in \mathcal{G}^A_\ell$ s.t. $\mathsf{idx_a}(g) = k$, with $(w^1_\iota, w^2_\iota, w_o) = \mathsf{wires}(g)$:
17: $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$.add($\mathsf{sid}_{w_o}$, [$\mathsf{sid}_{w^1_\iota}$, $\mathsf{sid}_{w^2_\iota}$], [w^1_ι.const, w^2_ι.const]).
18: Wait until $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$.Sh.receive($\mathsf{sid}_{w_o}$).

Multiplicative Evaluation
19: Upon Mult_Eval($\ell \in [1 : D]$):
20: Let $\vec{g} = [g_1, \ldots, g_{K_\ell}]$ be the multiplicative gates of ℓ-th layer of $\mathcal{C}$, ordered by $\mathsf{idx_m}$.
21: Let $(k_{\min}, k_{\max}) = (\mathsf{idx_m}(g_1), \mathsf{idx_m}(g_{K_\ell}))$
22: For each $\vec{g}$'s element g:
23: Let $(w^1_\iota, w^2_\iota, w_o) = \mathsf{wires}(g)$. Let $k = \mathsf{idx_m}(g)$.
24: $(\mathbf{sid}_x[k], \mathbf{sid}_y[k], \mathbf{sid}_z[k]) \leftarrow (\mathsf{sid}_{(w^1_\iota,k)}, \mathsf{sid}_{(w^2_\iota,k)}, \mathsf{sid}_{(w_o,k)})$
25: $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle$.(mult, $\mathbf{sid}_z, \mathbf{sid}_x, \mathbf{sid}_y, \mathbf{sid}_a, \mathbf{sid}_b, \mathbf{sid}_c$) ▷ See the batched multiplication helper

Accountability
26: **upon notification** (detection, sid_{idx}, G, π) **from** $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle$ or $\mathcal{F}^{\text{acc}}_{\text{idx_acs}}$:
27: **trigger notification** (detection, sid, G, π) ▷ Detected equivocation

(BreakAgreement, sid, μ^*, $\boldsymbol{\pi}$) supplied by the adversary, the functionality records the partition map μ^*, marks the execution as broken, and notifies all parties with a delayed detection event identifying the set G_i and accompanying proof (line 33– 37).

The pseudocode for the circuit evaluation realization $\mathcal{P}^{\text{acc}}_{\text{ampc}}$ is given in Algorithm 4. The notation associated with the arithmetic circuit is given in Appendix A.

We now describe the realization.

Online Phase Description. Defined in the $(\mathcal{F}^{\text{acc}}_{\text{idx_acs}}, \mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle, \mathcal{F}^{\text{acc}}_{\text{bhacss}})$-hybrid model, the protocol $\mathcal{P}^{\text{acc}}_{\text{ampc}}$ (Algorithm 4) proceeds in four stages: offline preparation, input sharing, circuit evaluation layer by layer, and output reconstruction.

Offline phase. Processes first enable Beaver triple generation by invoking the functionality $\mathcal{F}^{\text{acc}}_{\text{btg}}$ (line 1), and later receive a batch of triples indexed by session identifiers (line 2). Each triple (a, b, c) is assigned either to a multiplicative gate g (with index $\mathsf{midx}(g)$), or to a random input gate. These triples guarantee that the multiplicative relation $c = a \cdot b$ over the field $\mathbb{F}$ can be realized consistently across all processes.

Input sharing. Each process p_i with local input x_i shares it via $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$ (line 4). Upon receiving shares (line 6), processes validate them through $\mathcal{F}^{\text{acc}}_{\text{idx_acs}}$. Once a core set Q of $n - t$ processes is agreed upon (line 7), the additive evaluation for the trivial layer L_0 is triggered (line 8). At this point, the valuation of every input wire is initialized consistently with the circuit semantics.

Layer-by-layer evaluation. For each multiplicative depth $\ell \in [1 : D-1]$ (line 9), processes first trigger the multiplication subroutine (line 10), then additive evaluation for all additive gates in the layer (line 11).

– In an *additive evaluation* (line 14), for each additive gate $g \in \mathcal{G}^A_\ell$ (line 16), the outgoing wire w_o must satisfy:

$$\mathsf{valuation}(w_o) = w^1_\iota.\mathsf{const} \odot_{\mathbb{F}} \mathsf{valuation}(w^1_\iota) \oplus_{\mathbb{F}} w^2_\iota.\mathsf{const} \odot_{\mathbb{F}} \mathsf{valuation}(w^2_\iota).$$

This is implemented by invoking $\mathcal{F}^{\text{acc}}_{\text{bhacss}}$.local_add (line 17), and the result is materialized once the new share is received (line 18).

– In a *multiplicative evaluation* (line 19), the multiplicative gates of the current layer are processed in order (lines 20–21). For each gate g, the semantics require

$$\mathsf{valuation}(w_o) = \mathsf{valuation}(w^1_\iota) \otimes_{\mathbb{F}} \mathsf{valuation}(w^2_\iota).$$

This is realized via Beaver triples by invoking the batched helper $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle$.mult (Functionality 6). Concretely, the helper first computes the masked differences $\tilde{A} = x - a$ and $\tilde{B} = y - b$ using local_add (lines 3–4), then reconstructs $(\tilde{A}, \tilde{B})$ in batch (lines 5–6), and computes the public products $\tilde{A}\tilde{B}$ locally (line 8). Finally, it derives the shared output $[xy]$ by combining $\tilde{A}\tilde{B}$ with the triple (a, b, c) through

$$x \otimes y = \big((x \ominus a) \otimes (y \ominus b)\big) \oplus \big(b \otimes (x \ominus a)\big) \oplus \big(a \otimes (y \ominus b)\big) \oplus c,$$

implemented as a final local_add call (line 9).

Output reconstruction. Once all layers are processed, each process helps reconstruct its designated output wires (line 12), ensuring that the valuation of output wires.

Accountability. If any equivocation or inconsistency is detected, a detection notification (detection, sid, G, π) is triggered (lines 26–27), identifying the misbehaving processes and enabling exposure of at least $n - 2t$ of them in degraded mode.

Theorem 5 (Online-Phase). *(Theorem 2) Protocol $\mathcal{P}^{\text{acc}}_{\text{ampc}}$ UC-realizes the functionality $\mathcal{F}^{\text{acc}}_{\text{ampc}}$ in the $(\mathcal{F}^{\text{acc}}_{\text{bhacss}}, \mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc}}_{\text{bhacss}}\rangle, \mathcal{F}^{\text{acc}}_{\text{idx-acs}})$-hybrid model.*

The proof is postponed to Appendix B.

References

[1] R. Cramer, I. Damgård, and J. B. Nielsen, *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015. [Online]. Available: http://www.cambridge.org/de/academic/subjects/computer-science/cryptography-cryptology-and-coding/secure-multiparty-computation-and-secret-sharing?format=HB&isbn=9781107043053

[2] D. Evans, V. Kolesnikov, and M. Rosulek, "A pragmatic introduction to secure multi-party computation," *Found. Trends Priv. Secur.*, vol. 2, no. 2-3, pp. 70–246, 2018. [Online]. Available: https://doi.org/10.1561/3300000019

[3] R. Cleve, "Limits on the security of coin flips when half the processors are faulty (extended abstract)," in *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, J. Hartmanis, Ed. ACM, 1986, pp. 364–369. [Online]. Available: https://doi.org/10.1145/12130.12168

[4] C. Baum, N. Melissaris, R. Rachuri, and P. Scholl, "Cheater identification on a budget: MPC with identifiable abort from pairwise macs," in *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII*, ser. Lecture Notes in Computer Science, L. Reyzin and D. Stebila, Eds., vol. 14927. Springer, 2024, pp. 454–488. [Online]. Available: https://doi.org/10.1007/978-3-031-68397-8_14

[5] C. Baum, E. Orsini, and P. Scholl, "Efficient secure multiparty computation with identifiable abort," in *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, ser. Lecture Notes in Computer Science, M. Hirt and A. D. Smith, Eds., vol. 9985, 2016, pp. 461–490. [Online]. Available: https://doi.org/10.1007/978-3-662-53641-4_18

[6] C. Baum, E. Orsini, P. Scholl, and E. Soria-Vazquez, "Efficient constant-round MPC with identifiable abort and public verifiability," in *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, ser. Lecture Notes in Computer Science, D. Micciancio and T. Ristenpart, Eds., vol. 12171. Springer, 2020, pp. 562–592. [Online]. Available: https://doi.org/10.1007/978-3-030-56880-1_20

[7] M. Ciampi, D. Ravi, L. Siniscalchi, and H. Waldner, "Round-optimal multi-party computation with identifiable abort," in *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, ser. Lecture Notes in Computer Science, O. Dunkelman and S. Dziembowski, Eds., vol. 13275. Springer, 2022, pp. 335–364. [Online]. Available: https://doi.org/10.1007/978-3-031-06944-4_12

[8] R. Cohen, J. Doerner, Y. Kondi, and A. Shelat, "Secure multiparty computation with identifiable abort via vindicating release," in *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII*, ser. Lecture Notes in Computer Science, L. Reyzin and D. Stebila, Eds., vol. 14927. Springer, 2024, pp. 36–73. [Online]. Available: https://doi.org/10.1007/978-3-031-68397-8_2

[9] R. Cohen and Y. Lindell, "Fairness versus guaranteed output delivery in secure multiparty computation," in *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II*, ser. Lecture Notes in Computer Science, P. Sarkar and T. Iwata, Eds., vol. 8874. Springer, 2014, pp. 466–485. [Online]. Available: https://doi.org/10.1007/978-3-662-45608-8_25

[10] ——, "Fairness versus guaranteed output delivery in secure multiparty computation," *J. Cryptol.*, vol. 30, no. 4, pp. 1157–1186, 2017. [Online]. Available: https://doi.org/10.1007/s00145-016-9245-5

[11] R. K. Cunningham, B. Fuller, and S. Yakoubov, "Catching MPC cheaters: Identification and openability," in *Information Theoretic Security - 10th International Conference, ICITS 2017, Hong Kong, China, November 29 - December 2, 2017, Proceedings*, ser. Lecture Notes in Computer Science, J. Shikata, Ed., vol. 10681. Springer, 2017, pp. 110–134. [Online]. Available: https://doi.org/10.1007/978-3-319-72089-0_7

[12] Y. Ishai, R. Ostrovsky, and V. Zikas, "Secure multi-party computation with identifiable abort," in *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, ser. Lecture Notes in Computer Science, J. A. Garay and R. Gennaro, Eds., vol. 8617. Springer, 2014, pp. 369–386. [Online]. Available: https://doi.org/10.1007/978-3-662-44381-1_21

[13] G. Spini and S. Fehr, "Cheater detection in SPDZ multiparty computation," in *Information Theoretic Security - 9th International Conference, ICITS 2016, Tacoma, WA, USA, August 9-12, 2016, Revised Selected Papers*, ser. Lecture Notes in Computer Science, A. C. A. Nascimento and P. S. L. M. Barreto, Eds., vol. 10015, 2016, pp. 151–176. [Online]. Available: https://doi.org/10.1007/978-3-319-49175-2_8

[14] Y. Aumann and Y. Lindell, "Security against covert adversaries: Efficient protocols for realistic adversaries," in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, ser. Lecture Notes in Computer Science, S. P. Vadhan, Ed., vol. 4392. Springer, 2007, pp. 137–156. [Online]. Available: https://doi.org/10.1007/978-3-540-70936-7_8

[15] ——, "Security against covert adversaries: Efficient protocols for realistic adversaries," *J. Cryptol.*, vol. 23, no. 2, pp. 281–343, 2010. [Online]. Available: https://doi.org/10.1007/s00145-009-9040-7

[16] M. Rivinius, P. Reisert, D. Rausch, and R. Küsters, "Publicly accountable robust multi-party computation," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 2430–2449. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833608

[17] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous Secure Computations with Optimal Resilience," in *Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing*, 1994, pp. 183–192.

[18] M. Backes, F. Bendun, A. Choudhury, and A. Kate, "Asynchronous MPC with a strict honest majority using non-equivocation," in *ACM Symposium on Principles of Distributed Computing, PODC '14, Paris, France, July 15-18, 2014*, M. M. Halldórsson and S. Dolev, Eds. ACM, 2014, pp. 10–19. [Online]. Available: https://doi.org/10.1145/2611462.2611490

[19] R. Bhadauria and A. Choudhury, "Brief announcement: Asynchronous secure distributed computing with transferrable non-equivocation revisited," in *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC 2018, Egham, United Kingdom, July 23-27, 2018*, C. Newport and I. Keidar, Eds. ACM, 2018, pp. 265–267. [Online]. Available: https://dl.acm.org/citation.cfm?id=3212777

[20] S. H. Kamp and J. B. Nielsen, "Byzantine agreement decomposed: Honest majority asynchronous total-order broadcast from reliable broadcast," *IACR Cryptol. ePrint Arch.*, p. 1738, 2023. [Online]. Available: https://eprint.iacr.org/2023/1738

[21] M. Leinweber and H. Hartenstein, "Brief announcement: Let it TEE: asynchronous byzantine atomic broadcast with n $\geq$ 2f+1," in *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, ser. LIPIcs, R. Oshman, Ed., vol. 281. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 43:1–43:7. [Online]. Available: https://doi.org/10.4230/LIPIcs.DISC.2023.43

[22] M. Correia, G. S. Veronese, and L. C. Lung, "Asynchronous byzantine consensus with 2f+1 processes," in *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, S. Y. Shin, S. Ossowski, M. Schumacher, M. J. Palakal, and C. Hung, Eds. ACM, 2010, pp. 475–480. [Online]. Available: https://doi.org/10.1145/1774088.1774187

[23] R. Cohen, "Asynchronous secure multiparty computation in constant time," in *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, ser. Lecture Notes in Computer Science, C. Cheng, K. Chung, G. Persiano, and B. Yang, Eds., vol. 9615. Springer, 2016, pp. 183–207. [Online]. Available: https://doi.org/10.1007/978-3-662-49387-8_8

[24] G. Bracha and S. Toueg, "Resilient consensus protocols," in *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, R. L. Probert, N. A. Lynch, and N. Santoro, Eds. ACM, 1983, pp. 12–26. [Online]. Available: https://doi.org/10.1145/800221.806706

[25] R. Küsters, T. Truderung, and A. Vogt, "Accountability: definition and relationship to verifiability," in *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds. ACM, 2010, pp. 526–535. [Online]. Available: https://doi.org/10.1145/1866307.1866366

[26] P. Civit, S. Gilbert, V. Gramoli, R. Guerraoui, and J. Komatovic, "As easy as ABC: optimal (a)ccountable (b)yzantine (c)onsensus is easy!" in *2022 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2022, Lyon, France, May 30 - June 3, 2022*. IEEE, 2022, pp. 560–570. [Online]. Available: https://doi.org/10.1109/IPDPS53621.2022.00061

[27] ——, "As easy as ABC: optimal (a)ccountable (b)yzantine (c)onsensus is easy!" *J. Parallel Distributed Comput.*, vol. 181, p. 104743, 2023. [Online]. Available: https://doi.org/10.1016/j.jpdc.2023.104743

[28] P. Civit, D. Collins, V. Gramoli, R. Guerraoui, J. Komatovic, M. Vidigueira, and P. Zarbafian, "Scalable accountable byzantine agreement and beyond," in *IEEE Symposium on Security and Privacy, SP 2026 (To appear), San Francisco, CA, USA, May 18-20, 2026*. IEEE, 2026. [Online]. Available: https://eprint.iacr.org/2025/1277

[29] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 2139. Springer, 2001, pp. 524–541. [Online]. Available: https://doi.org/10.1007/3-540-44647-8_31

[30] B. Chor and L. Moscovici, "Solvability in Asynchronous Environments (Extended Abstract)," in *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 422–427. [Online]. Available: https://doi.org/10.1109/SFCS.1989.63513

[31] S. Das, S. Duan, S. Liu, A. Momose, L. Ren, and V. Shoup, "Asynchronous consensus without trusted setup or public-key cryptography," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds. ACM, 2024, pp. 3242–3256. [Online]. Available: https://doi.org/10.1145/3658644.3670327

[32] G. Bracha, "An Asynchronous [(n-1)/3]-Resilient Consensus Protocol," in *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, T. Kameda, J. Misra, J. G. Peters, and N. Santoro, Eds. ACM, 1984, pp. 154–162. [Online]. Available: https://doi.org/10.1145/800222.806743

[33] R. Canetti, Y. Lindell, R. Ostrovsky, and A. Sahai, "Universally composable two-party and multi-party secure computation," in *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, J. H. Reif, Ed. ACM, 2002, pp. 494–503. [Online]. Available: https://doi.org/10.1145/509907.509980

[34] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: practical accountability for distributed systems," in *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, T. C. Bressoud and M. F. Kaashoek, Eds. ACM, 2007, pp. 175–188. [Online]. Available: https://doi.org/10.1145/1294261.1294279

[35] A. Haeberlen, I. C. Avramopoulos, J. Rexford, and P. Druschel, "Netreview: Detecting when interdomain routing goes wrong," in *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*, J. Rexford and E. G. Sirer, Eds. USENIX Association, 2009, pp. 437–452. [Online]. Available: http://www.usenix.org/events/nsdi09/tech/full_papers/haeberlen/haeberlen.pdf

[36] M. Backes, P. Druschel, A. Haeberlen, and D. Unruh, "CSAR: A practical and provable technique to make randomized systems accountable," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*. The Internet Society, 2009. [Online]. Available: https://www.ndss-symposium.org/ndss2009/csar-practical-and-provable-technique-make-randomized-systems-accountable/

[37] F. C. Freiling, R. Guerraoui, and P. Kuznetsov, "The failure detector abstraction," *ACM Comput. Surv.*, vol. 43, no. 2, pp. 9:1–9:40, 2011. [Online]. Available: https://doi.org/10.1145/1883612.1883616

[38] P. Civit, D. Collins, V. Gramoli, R. Guerraoui, J. Komatovic, M. Vidigueira, and P. Zarbafian, "Computing in a safe house: Accountable universally composable asynchronous secure distributed computing," Cryptology ePrint Archive, Paper 2026/182, 2026. [Online]. Available: https://eprint.iacr.org/2026/182

[39] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, A. V. Aho, Ed. ACM, 1987, pp. 218–229. [Online]. Available: https://doi.org/10.1145/28395.28420

[40] C. Ganesh, Y. Kondi, C. Orlandi, M. Pancholi, A. Takahashi, and D. Tschudi, "Witness-succinct universally-composable snarks," in *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, ser. Lecture Notes in Computer Science, C. Hazay and M. Stam, Eds., vol. 14005. Springer, 2023, pp. 315–346. [Online]. Available: https://doi.org/10.1007/978-3-031-30617-4_11

[41] A. Bandarupalli, X. Ji, A. Kate, C. Liu-Zhang, and Y. Song, "Computationally efficient asynchronous MPC with linear communication and low additive overhead," in *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, Y. T. Kalai and S. F. Kamara, Eds. Springer, 2025, pp. 261–294. [Online]. Available: https://doi.org/10.1007/978-3-032-01884-7_9

[42] S. Das, Z. Xiang, A. Tomescu, A. Spiegelman, B. Pinkas, and L. Ren, "Verifiable secret sharing simplified," in *IEEE Symposium on Security and Privacy, SP 2025, San Francisco, CA, USA, May 12-15, 2025*, M. Blanton, W. Enck, and C. Nita-Rotaru, Eds. IEEE, 2025, pp. 633–651. [Online]. Available: https://doi.org/10.1109/SP61157.2025.00046

[43] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, ser. Lecture Notes in Computer Science, J. Feigenbaum, Ed., vol. 576. Springer, 1991, pp. 420–432. [Online]. Available: https://doi.org/10.1007/3-540-46766-1_34

[44] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern, "Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation," in *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, ser. Lecture Notes in Computer Science, H. Handschuh and A. Lysyanskaya, Eds., vol. 14081. Springer, 2023, pp. 39–70. [Online]. Available: https://doi.org/10.1007/978-3-031-38557-5_2

[45] N. Alhaddad, M. Varia, and Z. Yang, "Haven++: Batched and packed dual-threshold asynchronous complete secret sharing with applications," *IACR Commun. Cryptol.*, vol. 1, no. 4, p. 3, 2024. [Online]. Available: https://doi.org/10.62056/a0qj5w7sf

[46] J. A. Garay, P. D. MacKenzie, and K. Yang, "Strengthening zero-knowledge protocols using signatures," in *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings*, ser. Lecture Notes in Computer Science, E. Biham, Ed., vol. 2656. Springer, 2003, pp. 177–194. [Online]. Available: https://doi.org/10.1007/3-540-39200-9_11

[47] M. Chen, P. Dey, C. Ganesh, P. Mukherjee, P. Sarkar, and S. Sasmal, "Universally composable non-interactive zero-knowledge from sigma protocols via a new straight-line compiler," in *Public-Key Cryptography - PKC 2025 - 28th IACR International Conference on Practice and Theory of Public-Key Cryptography, Røros, Norway, May 12-15, 2025, Proceedings, Part I*, ser. Lecture Notes in Computer Science, T. Jager and J. Pan, Eds., vol. 15674. Springer, 2025, pp. 381–417. [Online]. Available: https://doi.org/10.1007/978-3-031-91820-9_13

[48] Z. Beerliová-Trubíniová and M. Hirt, "Perfectly-secure MPC with linear communication complexity," in *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, ser. Lecture Notes in Computer Science, R. Canetti, Ed., vol. 4948. Springer, 2008, pp. 213–230. [Online]. Available: https://doi.org/10.1007/978-3-540-78524-8_13

[49] M. Graf, R. Küsters, and D. Rausch, "AUC: accountable universal composability," in *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 1148–1167. [Online]. Available: https://doi.org/10.1109/SP46215.2023.10179384

[50] I. Damgård and J. B. Nielsen, "Scalable and unconditionally secure multiparty computation," in *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, ser. Lecture Notes in Computer Science, A. Menezes, Ed., vol. 4622. Springer, 2007, pp. 572–590. [Online]. Available: https://doi.org/10.1007/978-3-540-74143-5_32

[51] M. K. Franklin and S. Haber, "Joint encryption and message-efficient secure computation," in *Advances in Cryptology - CRYPTO '93, 13th Annual International Cryptology Conference, Santa Barbara, California, USA, August 22-26, 1993, Proceedings*, ser. Lecture Notes in Computer Science, D. R. Stinson, Ed., vol. 773. Springer, 1993, pp. 266–277. [Online]. Available: https://doi.org/10.1007/3-540-48329-2_23

[52] R. Cramer, I. Damgård, and J. B. Nielsen, "Multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, ser. Lecture Notes in Computer Science, B. Pfitzmann, Ed., vol. 2045. Springer, 2001, pp. 280–299. [Online]. Available: https://doi.org/10.1007/3-540-44987-6_18

[53] I. Damgård and J. B. Nielsen, "Universally composable efficient multiparty computation from threshold homomorphic encryption," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 247–264. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_15

[54] B. Chevallier-Mames, P. Paillier, and D. Pointcheval, "Encoding-free elgamal encryption without random oracles," in *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, ser. Lecture Notes in Computer Science, M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, Eds., vol. 3958. Springer, 2006, pp. 91–104. [Online]. Available: https://doi.org/10.1007/11745853_7

[55] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. Miller, "hbacss: How to robustly share many secrets," in *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/auto-draft-245/

[56] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings*, ser. Lecture Notes in Computer Science, M. Abe, Ed., vol. 6477. Springer, 2010, pp. 177–194. [Online]. Available: https://doi.org/10.1007/978-3-642-17373-8_11

[57] E. Fujisaki, "Improving practical uc-secure commitments based on the DDH assumption," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 105-A, no. 3, pp. 182–194, 2022. [Online]. Available: https://doi.org/10.1587/transfun.2021cip0014

[58] L. Braun, G. Castagnos, I. Damgård, F. Laguillaumie, K. Melissaris, C. Orlandi, and I. Tucker, "An improved threshold homomorphic cryptosystem based on class groups," *Cryptogr. Commun.*, vol. 17, no. 5, pp. 1367–1406, 2025. [Online]. Available: https://doi.org/10.1007/s12095-025-00826-2

[59] Z. Brakerski, O. Friedman, A. Marmor, D. Mutzari, Y. Spiizer, and N. Trieu, "Threshold FHE with efficient asynchronous decryption," *IACR Cryptol. ePrint Arch.*, p. 712, 2025. [Online]. Available: https://eprint.iacr.org/2025/712

[60] J. Camenisch, S. Krenn, R. Küsters, and D. Rausch, "iuc: Flexible universal composability made simple," in *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part III*, ser. Lecture Notes in Computer Science, S. D. Galbraith and S. Moriai, Eds., vol. 11923. Springer, 2019, pp. 191–221. [Online]. Available: https://doi.org/10.1007/978-3-030-34618-8_7

[61] R. Küsters, M. Tuengerthal, and D. Rausch, "The IITM model: A simple and expressive model for universal composability," *J. Cryptol.*, vol. 33, no. 4, pp. 1461–1584, 2020. [Online]. Available: https://doi.org/10.1007/s00145-020-09352-1

[62] D. Rausch, R. Küsters, and C. Chevalier, "Embedding the UC model into the IITM model," in *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part II*, ser. Lecture Notes in Computer Science, O. Dunkelman and S. Dziembowski, Eds., vol. 13276. Springer, 2022, pp. 242–272. [Online]. Available: https://doi.org/10.1007/978-3-031-07085-3_9

[63] R. Canetti, Y. Dodis, R. Pass, and S. Walfish, "Universally composable security with global setup," in *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, ser. Lecture Notes in Computer Science, S. P. Vadhan, Ed., vol. 4392. Springer, 2007, pp. 61–85. [Online]. Available: https://doi.org/10.1007/978-3-540-70936-7_4

[64] R. Canetti and T. Rabin, "Universal composition with joint state," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 265–281. [Online]. Available: https://doi.org/10.1007/978-3-540-45146-4_16

[65] R. Canetti, "Universally composable signature, certification, and authentication," in *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 2004, p. 219. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CSFW.2004.24

[66] M. Backes, D. Hofheinz, J. Müller-Quade, and D. Unruh, "On fairness in simulatability-based cryptographic systems," in *Proceedings of the 2005 ACM workshop on Formal methods in security engineering, FMSE 2005, Fairfax, VA, USA, November 11, 2005*, V. Atluri, P. Samarati, R. Küsters, and J. C. Mitchell, Eds. ACM, 2005, pp. 13–22. [Online]. Available: https://doi.org/10.1145/1103576.1103579

[67] R. Canetti, "Universally composable security: A new paradigm for cryptographic protocols," in *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*. IEEE Computer Society, 2001, pp. 136–145. [Online]. Available: https://doi.org/10.1109/SFCS.2001.959888

[68] M. Backes, B. Pfitzmann, M. Steiner, and M. Waidner, "Polynomial fairness and liveness," in *15th IEEE Computer Security Foundations Workshop (CSFW-15 2002), 24-26 June 2002, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society, 2002, pp. 160–174. [Online]. Available: https://doi.org/10.1109/CSFW.2002.1021814

[69] M. Backes, B. Pfitzmann, and M. Waidner, "Polynomial liveness," *J. Comput. Secur.*, vol. 12, no. 3-4, pp. 589–617, 2004. [Online]. Available: https://doi.org/10.3233/jcs-2004-123-409

[70] N. A. Lynch, *Distributed Algorithms*. Elsevier, 1996.

[71] J. Groth and V. Shoup, "Fast batched asynchronous distributed key generation," in *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*, ser. Lecture Notes in Computer Science, M. Joye and G. Leander, Eds., vol. 14655. Springer, 2024, pp. 370–400. [Online]. Available: https://doi.org/10.1007/978-3-031-58740-5_13

[72] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract)," in *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, G. Neiger, Ed. ACM, 2000, pp. 123–132. [Online]. Available: https://doi.org/10.1145/343477.343531

[73] ——, "Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography," *J. Cryptol.*, vol. 18, no. 3, pp. 219–246, 2005. [Online]. Available: https://doi.org/10.1007/s00145-005-0318-0

[74] V. Shoup, "A theoretical take on a practical consensus protocol," *Cryptology ePrint Archive*, 2024.

[75] V. Shoup and N. P. Smart, "Lightweight asynchronous verifiable secret sharing with optimal resilience," *J. Cryptol.*, vol. 37, no. 3, p. 27, 2024. [Online]. Available: https://doi.org/10.1007/s00145-024-09505-6

[76] N. Alhaddad, S. Das, S. Duan, L. Ren, M. Varia, Z. Xiang, and H. Zhang, "Balanced byzantine reliable broadcast with near-optimal communication and improved computation," in *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, A. Milani and P. Woelfel, Eds. ACM, 2022, pp. 399–417. [Online]. Available: https://doi.org/10.1145/3519270.3538475

Appendix Roadmap

Appendix A	Circuit-evaluation conventions	p. 18
Appendix B	UC proof of the online phase	p. 19
Appendix C	UC proof of the linear APBHACSS realization	p. 20
Appendix D	Coupled shared randomness	p. 22
Appendix E	Beaver triples migration	p. 27
Appendix F	Beaver triples generation	p. 29
Appendix G	Homomorphic (trapdoor) polynomial commitment	p. 33
Appendix H	Universal Composability	p. 37
Appendix I	Easily accountable functionalities	p. 37
Appendix J	Judicial reports and evidence handling	p. 41

Appendix A.
Circuit Evaluation

The arithmetic circuit $\mathcal{C}$ is modeled as a labeled directed acyclic graph $(\mathcal{G}, \mathcal{W}, \mathcal{L})$, where $\mathcal{G}$ is the set of gates (vertices), $\mathcal{W}$ is the set of wires (edges), and $\mathcal{L} : \mathcal{G} \cup \mathcal{W} \to \{0,1\}^*$ is a labeling function that may attach arbitrary auxiliary information to gates and wires. Each wire $w = (g, g')$ connects an output of gate g to an input of gate g'. A gate has at most two incoming wires and at most one outgoing wire.

Gate types. An *input gate* g has no incoming wires and one outgoing wire, denoted by $\mathsf{wire}(g)$. An *output gate* g has one incoming wire, denoted by $\mathsf{wire}(g)$, and no outgoing wire; the number of output gates (typically proportional to n) is denoted by c_O. An *intermediate gate* has two incoming wires w_ι^1, w_ι^2 and one outgoing wire w_o; we note $\mathsf{wires}(g) = (w_\iota^1, w_\iota^2, w_o)$. Intermediate gates are partitioned into *additive gates* and *multiplicative gates*. The set of multiplicative gates and additive gates are denoted by $\mathcal{G}^M$ and $\mathcal{G}^A$, respectively. We note $c_M = |\mathcal{G}^M|$ and $c_A = |\mathcal{G}^A|$.

Wire types. *Input wires* are the outgoing wires of input gates; *output wires* are the incoming wires of output gates; and *intermediate wires* are all other wires. Input wires are further classified as *process input wires*, carrying inputs supplied by the processes, whose number (typically proportional to n) is denoted by c_I; *constant input wires*, labeled with a fixed element of R; and *random input wires*, initialized with uniformly random elements of R at the start of evaluation, whose number is denoted by c_R.

During the online phase, each wire is associated with a (secretly shared) value of R, specified by a valuation function $\mathsf{valuation} : \mathcal{W} \to R$. This valuation must be consistent with the semantics of the gates: for an additive gate g with incoming wires w_ι^1, w_ι^2, outgoing wire w_o, and constants $w_\iota^1.\mathsf{const}, w_\iota^2.\mathsf{const}$,

$$\mathsf{valuation}(w_o) \;=\; w_\iota^1.\mathsf{const} \odot_{\mathbb{F}} \mathsf{valuation}(w_\iota^1) \oplus_{\mathbb{F}}$$

$$w_\iota^2.\mathsf{const} \odot_{\mathbb{F}} \mathsf{valuation}(w_\iota^2).$$

For a multiplicative gate g with incoming wires w_ι^1, w_ι^2 and outgoing wire w_o,

$$\mathsf{valuation}(w_o) \;=\; \mathsf{valuation}(w_\iota^1) \otimes_R \mathsf{valuation}(w_\iota^2).$$

Finally, valuation is said to be consistent with $\mathcal{C}$ if it is consistent with all intermediate gates of $\mathcal{C}$.

Define the *multiplicative depth* $\mathsf{md} : \mathcal{G} \to \mathbb{N}$ by $\mathsf{md}(g) :=$

$$\begin{cases} 0, & \text{if } g \text{ is an input gate,} \\ \max\limits_{(u,g)\in\mathcal{W}} \mathsf{md}(u), & \text{if } g \text{ is additive, or output,} \\ \max\limits_{(u,g)\in\mathcal{W}} \mathsf{md}(u) \;+\; 1, & \text{if } g \text{ is multiplicative.} \end{cases}$$

For $d \in \mathbb{N}$, the *d-th layer* is the labeled subgraph

$$L_d \;:=\; \big(\; \mathcal{G}_d,\; \mathcal{W}_d,\; \mathcal{L}\restriction_{\mathcal{G}_d \cup \mathcal{W}_d} \;\big), \text{ where}$$

$$\mathcal{G}_d := \{g \in \mathcal{G} : \mathsf{md}(g) = d\},$$

$$\mathcal{W}_d := \{(u,v) \in \mathcal{W} : u \in \mathcal{G}_d \text{ or } v \in \mathcal{G}_d\}.$$

These layers satisfy: (1) *Uniqueness*: $\{L_d\}_{d\geq 0}$ is uniquely determined by $\mathcal{C}$; (2) *Partition*: the sets $\{\mathcal{G}_d\}_{d\geq 0}$ form a disjoint partition of $\mathcal{G}$; (3) *Independency*: if $g \in \mathcal{G}_d$ is multiplicative and $(u,g) \in \mathcal{W}$, then $\mathsf{md}(u) \leq d-1$; in particular, no incoming wire of a multiplicative gate originates in the same layer; (4) *Trivial initial layer*: L_0 contains no multiplicative gates (it may contain input, and additive gates); we call L_0 the *trivial layer*; and (5) *Maximality*: each L_d is maximal among subgraphs satisfying items (2)–(3), since adding any gate or wire violates the definition.

The *multiplicative depth* of $\mathcal{C}$ is $D := \max_{g\in\mathcal{G}} \mathsf{md}(g)$. Each circuit is associated with two bijections $\mathsf{idx}_\mathsf{m} : \mathcal{G}^M \to [1..c_M]$ and $\mathsf{idx}_\mathsf{a} : \mathcal{G}^A \to [1..c_A]$, such that $\forall g_m, g'_m \in \mathcal{G}^M$, $\mathsf{idx}_\mathsf{m}(g_m) < \mathsf{idx}_\mathsf{m}(g'_m)$ if $\mathsf{md}(g_m) < \mathsf{md}(g'_m)$, and $\forall g_a, g'_a \in \mathcal{G}^A$ $\mathsf{idx}_\mathsf{a}(g_a) < \mathsf{idx}_\mathsf{a}(g'_a)$ if $\mathsf{md}(g_a) < \mathsf{md}(g'_a)$ or $\mathsf{md}(g_a) = \mathsf{md}(g'_a)$ and there exists a path from g_a to g'_a.

Appendix B.
Proof of Theorem 5

Here we provide the missing proof of Theorem 5.

Proof. Fix any real-world adversary $\mathcal{A}$. We construct a simulator $\mathcal{S}$ such that no environment $\mathcal{E}$ can tell the real execution from the ideal one.

Simulator. The simulator internally runs a full simulation of the real-world execution, including a simulated adversary $\mathcal{A}^{\text{sim}}$, and forwards all messages between $\mathcal{A}^{\text{sim}}$ and the environment $\mathcal{E}$ *verbatim*. Whenever $\mathcal{A}^{\text{sim}}$ sends a BreakLiveness command, $\mathcal{S}$ forwards the same command to $\mathcal{F}_{\text{ampc}}^{\text{acc}}$. $\mathcal{S}$ maintains the partitions map $\mu_{\mathcal{S}}^{acs}, \mu_{\mathcal{S}}^{in}, \mu_{\mathcal{S}}^{btg}, \mu_{\mathcal{S}}^{*} : [n] \to [n]$, such that every time $\mu_{\mathcal{S}}^{acs}$ or $\mu_{\mathcal{S}}^{in}$ is updated, $\mathcal{S}$ updates $\mu_{\mathcal{S}}^{*}$ such that $\forall i, j \in [n]$: $\mu_{\mathcal{S}}^{*}(i) = \mu_{\mathcal{S}}^{*}(j) \iff \forall \tilde{\mu}_{\mathcal{S}} \in \{\mu_{\mathcal{S}}^{acs}, \mu_{\mathcal{S}}^{in}, \mu_{\mathcal{S}}^{btg}\}$, $\tilde{\mu}_{\mathcal{S}}(i) = \tilde{\mu}_{\mathcal{S}}(j)$

In the nominal case, the only deviations from an exact real execution are the following interface adjustments to connect the simulation with the ideal execution.

1) **(Simulation of honest inputs.)** Whenever $\mathcal{S}$ is notified that an honest ideal-world party p_i^D provided its input, the simulator instructs the simulated real-world party p_i^{sim} to invoke $\mathcal{F}_{\text{bacss}}^{\text{acc}}$ on $\boldsymbol{\iota}_i^{\text{sim}}$, which is a uniformly **random** placeholder input, rather than the true ideal-world input $\boldsymbol{\iota}_i^D$.
2) **(Simulation of corrupted outputs.)** Whenever the simulated $\mathcal{F}_{\text{bacss}}^{\text{acc}}$ is about to reveal the valuation of an output wire associated to a corrupted party, $\mathcal{S}$ **replaces** this valuation with the corresponding output received from $\mathcal{F}_{\text{ampc}}^{\text{acc}}$.

Break events. Whenever $\mathcal{A}^{\text{sim}}$ sends a valid $(\mathsf{BreakAgreement}, \mu, \boldsymbol{\pi})$ to $\mathcal{F}_{\text{idx-acs}}^{\text{acc}}$ (resp., $\mathcal{F}_{\text{bhacss}}^{\text{acc}}$, $\mathcal{F}_{\text{btg}}^{\text{acc}}$), $\mathcal{S}$ updates $\mu_{\mathcal{S}}^{acs}$ (resp., $\mu_{\mathcal{S}}^{in}$, $\mu_{\mathcal{S}}^{btg}$) with μ, updates $\mu_{\mathcal{S}}^{*}$ accordingly and sends to $\mathcal{F}_{\text{ampc}}^{\text{acc}}$ the command $(\mathsf{BreakAgreement}, \mu_{\mathcal{S}}^{*}, \boldsymbol{\pi})$.

Extraction of corrupted inputs. Whenever the simulated secret-sharing functionality $\mathcal{F}_{\text{bhacss}}^{\text{acc}}$ receives input $\boldsymbol{\iota}_j$ from the simulated real-world corrupted party p_j^{sim}, the simulator reads $\boldsymbol{\iota}_j$, and instructs the corresponding corrupted ideal-world party p_j^D to submit $\boldsymbol{\iota}_j$ to $\mathcal{F}_{\text{ampc}}^{\text{acc}}$.

Choosing the qualified set. When, in the simulated real execution, $\mathcal{F}_{\text{idx-acs}}^{\text{acc}}$ outputs a set Q^k to honest process p_i such that $\mu_{\mathcal{S}}^{*}(i) = k$, for the first time for this k, the simulator sends $(\mathsf{ChooseSet}, k, Q^k)$ to $\mathcal{F}_{\text{ampc}}^{\text{acc}}$, receives the outputs of corrupted processes in return and store them.

Honest outputs. Whenever a simulated honest party p_i^{sim} triggers $(\mathsf{Output}, \boldsymbol{o}_i^{\text{sim}})$, $\mathcal{S}$ instructs $\mathcal{F}_{\text{ampc}}^{\text{acc}}$ to send to ideal process p_i^D its output (unknown to $\mathcal{S}$).

Masks Reveal. Regardless of any partitioning, Beaver triples appear uniformly random on a per-partition basis to $\mathcal{A}^{\text{sim}}$. Consequently, the masks opened (line 6 in Functionality 6, referenced by line 25 in the protocol)are uniformly distributed within the corresponding partition.

Partitionwise correctness. We briefly justify the correctness of the computation performed by $\mathcal{F}_{\text{ampc}}^{\text{acc}}$ for each partition k (line 27). Fix any partition index k and a qualified set Q^k selected by the adversary. For every party $p_i \in Q^k$, let $\boldsymbol{x}_i^k$ denote the input value used by the functionality, and let $\boldsymbol{x}^k = (\boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_n^k)$ be the resulting input vector, with default inputs substituted for parties outside Q^k.

Additions are evaluated internally by the additively homomorphic $\mathcal{F}_{\text{bhacss}}^{\text{acc}}$ functionality and therefore preserve exact correctness over $\mathbb{F}$. For multiplication gates, correctness follows from the standard Beaver multiplication identity. Specifically, for any multiplication gate computing $z = x \otimes_{\mathbb{F}} y$, the functionality relies on a Beaver triple (a, b, c) with $c = a \otimes_{\mathbb{F}} b$, and locally reconstructs masked values $A = x \ominus_{\mathbb{F}} a$ and $B = y \ominus_{\mathbb{F}} b$. The resulting output is computed as

$$z \;=\; (A \otimes_{\mathbb{F}} B) \oplus_{\mathbb{F}} (b \otimes_{\mathbb{F}} A) \oplus_{\mathbb{F}} (a \otimes_{\mathbb{F}} B) \oplus_{\mathbb{F}} c,$$

which holds as an algebraic identity over $\mathbb{F}$, independently of how the triple (a, b, c) was generated or distributed.

Since this identity holds for every multiplication gate, and since all additions are computed exactly, the evaluation of the arithmetic circuit $\mathcal{C}$ is correct gate by gate. Therefore, for each partition k, the functionality outputs

$$(\boldsymbol{o}_1^k, \ldots, \boldsymbol{o}_n^k) \;=\; \mathcal{C}(\boldsymbol{x}_1^k, \ldots, \boldsymbol{x}_n^k),$$

establishing partitionwise correctness as claimed.

Indistinguishability argument. The only places where the simulated execution may differ from a real execution are in the nominal case:

(i) where honest parties' inputs are replaced by random ones. This does not change the view of $\mathcal{A}^{\text{sim}}$, since the only information he has are the uniformly random masks revealed by $\mathcal{F}_{\text{btg}}^{\text{acc}}$ (line 6 in Functionality 6, referenced by line 25 in the protocol).

(ii) where corrupted output wires are replaced by their valuations from the ideal simulation, which matches the ones of the real-world execution. Since these are the only value-dependent leakage, the view is indistinguishable.

Liveness. Liveness is guaranteed by the liveness of the underlying functionalities $\mathcal{F}_{\text{bhacss}}^{\text{acc}}$, $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{bhacss}}^{\text{acc}}\rangle$, and $\mathcal{F}_{\text{idx-acs}}^{\text{acc}}$.

Appendix C.
UC Proof of Theorem 4

Proof. Foreword. We consider the ideal functionality

$$\mathcal{F}_{\text{pbhacss}}^{\mathsf{AH,acc}} \;=\; p_1^D \mid \cdots \mid p_n^D \mid \tilde{\mathcal{F}}_{\mathsf{pbhacss}}^{\mathsf{AH,acc}},$$

where $\tilde{\mathcal{F}}_{\mathsf{pbhacss}}^{\mathsf{AH,acc}}$ denotes $\mathcal{F}_{\text{pbhacss}}^{\mathsf{AH,acc}}$ stripped of the dummy processes $p_1^D, \ldots, p_n^D$. We work in the $(\mathcal{F}_{\text{rbc}}^{\text{acc}}, \mathcal{F}_{\text{zkpok}}(\mathcal{R}_{\text{vepc}}^{(\boldsymbol{\alpha},\ell)}), \mathcal{F}_{\text{ack}})$-hybrid model and analyze the real protocol

$$\mathcal{P}_{\text{pbhacss}}^{\mathsf{AH,acc}} \;=\; \bar{p}_1 \mid \cdots \mid \bar{p}_n \mid \mathcal{F}_{\text{rbc}}^{\text{acc}} \mid \mathcal{F}_{\text{zkpok}}(\mathcal{R}_{\text{vepc}}^{(\boldsymbol{\alpha},\ell)}) \mid \mathcal{F}_{\text{ack}},$$

whose pseudocode is given in Algorithm 1.

Let $\mathcal{A}$ be an arbitrary real-world adversary for $\mathcal{P}_{\text{pbhacss}}^{\mathsf{AH,acc}}$. We construct a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\text{pbhacss}}^{\mathsf{AH,acc}}$ and the environment $\mathcal{E}$ so that $\mathcal{E}$ cannot distinguish the ideal execution from the real one. Simulator $\mathcal{S}$ internally runs a copy of $\mathcal{A}$, denoted $\mathcal{A}^{\text{sim}}$. For each process $\bar{p}_k$, $\mathcal{S}$ simulates a copy $\bar{p}_k^{\mathsf{sim}}$. Whenever $\bar{p}_k^{\mathsf{sim}}$ is corrupted by $\mathcal{A}^{\text{sim}}$, $\mathcal{S}$ corrupts the corresponding dummy process p_k^D in the ideal world.

Messages from $\mathcal{E}$ to $\mathcal{S}$ (resp., a corrupted dummy process p_k^D) are forwarded to $\mathcal{A}^{\text{sim}}$ (resp., $\bar{p}_k^{\text{sim}}$), and messages from $\mathcal{A}^{\text{sim}}$ (resp., $\bar{p}_k^{\text{sim}}$) to $\mathcal{E}$ are forwarded back verbatim by $\mathcal{S}$ (resp., p_k^D).

Trapdoor setup for polynomial commitments.. In the real world, the SRS for TPC and the CRS for TC are generated by $\mathcal{F}_{\text{srs}}^{\text{acc}}$. In the ideal world, $\mathcal{S}$ computes itself $(\mathsf{srs}, \tau) \leftarrow \mathsf{TPC.Gen}(1^\kappa)$, and $(\mathsf{srs}_0, \tau_0) \leftarrow \mathsf{TC.Gen}(1^\kappa)$ on behalf of the simulated functionality $\mathcal{F}_{\text{srs}}^{\text{acc}}$, and stores $(\mathsf{srs}, \mathsf{srs}_0)$.

(A) Simulated real-world honest dealer.. When sid corresponds to an honest dealer in the simulated real world, the simulator has no access to the true secret. Instead, it creates a *dummy* Shamir–PCS sharing using the trapdoor interfaces of TC and TPC.

For each column $j \in [n]$, it samples fresh trapdoor states and dummy commitments:

$$\big(\sigma^{*,\diamond}[j],\, c^{*,\diamond}[j]\big) \leftarrow \mathsf{TPC.TCom}^{\tau}(), \qquad \big(\sigma_{r_0}^{*,\diamond}[j],\, c_{r_0}^{*,\diamond}[j]\big) \leftarrow \mathsf{TPC.TCom}^{\tau}(), \text{and}$$

$$\big(\sigma_0^{*,\diamond}[j],\, c_0^{*,\diamond}[j]\big) \leftarrow \mathsf{TC.TCom}^{\tau_0}().$$

Then, for each $(i,j) \in [n]^2$, it computes dummy evaluations and openings using TPC.TOpen:

$$\big(\Phi^{*,\diamond}[i,j],\, \Theta^{*,\diamond}[i,j]\big) \leftarrow \mathsf{TPC.TOpen}_{\alpha_i}^{\tau}\big(c^{*,\diamond}[j],\, \sigma^{*,\diamond}[j]\big),$$

$$\big(\Phi_{r_0}^{*,\diamond}[i,j],\, \Theta_{r_0}^{*,\diamond}[i,j]\big) \leftarrow \mathsf{TPC.TOpen}_{\alpha_i}^{\tau}\big(c_{r_0}^{*,\diamond}[j],\, \sigma_{r_0}^{*,\diamond}[j]\big).$$

These values are then encrypted element-wise for distribution:

$$\hat{\Phi}^{*,\diamond}[i,j] := \mathsf{Enc}_{\mathsf{pk}_i}\big(\Phi^{*,\diamond}[i,j];\, *\big), \qquad \hat{\Theta}^{*,\diamond}[i,j] := \mathsf{Enc}_{\mathsf{pk}_i}\big(\Theta^{*,\diamond}[i,j];\, *\big),$$

$$\hat{\Phi}_{r_0}^{*,\diamond}[i,j] := \mathsf{Enc}_{\mathsf{pk}_i}\big(\Phi_{r_0}^{*,\diamond}[i,j];\, *\big), \qquad \hat{\Theta}_{r_0}^{*,\diamond}[i,j] := \mathsf{Enc}_{\mathsf{pk}_i}\big(\Theta_{r_0}^{*,\diamond}[i,j];\, *\big).$$

For each $j \in [n]$, the simulator prepares the statement

$$stmt^{*,\diamond}[j] = \big(c_0^{*,\diamond}[j], c^{*,\diamond}[j], \hat{\Theta}^{*,\diamond}[:,j], \hat{\Phi}^{*,\diamond}[:,j], c_{r_0}^{*,\diamond}[j], \hat{\Theta}_{r_0}^{*,\diamond}[:,j], \hat{\Phi}_{r_0}^{*,\diamond}[:,j]\big),$$

which it writes directly on the *verifier tapes* of the ideal functionalities:

$$(c_0^{*,\diamond}, c^{*,\diamond}, c_{r_0}^{*,\diamond}) \rightarrow\; \mathcal{F}_{\text{rbc}}^{\text{acc}}, \qquad \big(stmt^{*,\diamond}[j]\big)_{j\in[n]} \rightarrow\; \mathcal{F}_{\text{zkPoK}}(R_{vepc}).$$

Correct processes then execute the real-world protocol faithfully, based on this dummy material.

Finally, the simulator applies the linear inverse encoding $\mathsf{Decode}_{(\boldsymbol{\alpha},[K])}$ to all dummy vectors and, for each $j \in [n]$, stores in $material[\mathsf{sid}_j]$:

$$\big(\sigma_0^{\diamond}[j],\, c_0^{\diamond}[j], \sigma^{\diamond}[j],\, c^{\diamond}[j], \Theta^{\diamond}[:,j],\, \Phi^{\diamond}[:,j], \sigma_{r_0}^{\diamond}[j],\, c_{r_0}^{\diamond}[j], \Theta_{r_0}^{\diamond}[:,j],\, \Phi_{r_0}^{\diamond}[:,j]\big),$$

where

$$x^{\diamond} := \mathsf{Decode}_{(\boldsymbol{\alpha},[K])}\big(x^{*,\diamond}\big).$$

(B) Simulated real-world corrupted dealer.. If a sharing session sid led by a corrupted dealer $\bar{p}_d^{sim}$ in the simulated real-world is *acknowledged* by a subset of correct simulated processes, the simulator can read the prover's tapes of $\mathcal{F}_{\mathrm{zkPoK}}$ and extract the corresponding unique pair of sharing polynomials $(\boldsymbol{\Xi}, \boldsymbol{\Xi}_{r_0})$.

It then:

- order p_d^D to send a share request for $\boldsymbol{\Xi}[1,:]$ to the $\mathcal{F}_{\mathrm{pbhacss}}^{\mathsf{AH,acc}}$ functionality.
- applies TPC.Share to each column of $\boldsymbol{\Xi}$ and $\boldsymbol{\Xi}_{r_0}$ to obtain the honest sharing material $(\mathbf{c}_0^\diamond,\ \mathbf{c}^\diamond,\ \boldsymbol{\Theta}^\diamond,\ \boldsymbol{\Phi}^\diamond, \mathbf{c}_{r_0}^\diamond,\ \boldsymbol{\Theta}_{r_0}^\diamond,\ \boldsymbol{\Phi}_{r_0}^\diamond)$;
- applies TPC.TSim to each column of $\boldsymbol{\Xi}$ and $\boldsymbol{\Xi}_{r_0}$, and TC.TSim to each pair $(\boldsymbol{\Xi}[1,j], \boldsymbol{\Xi}_{r_0}[1,j])$, thereby deriving a simulation state $(\sigma^\diamond[j], \sigma_{r_0}^\diamond[j], \sigma_0^\diamond[j])$ such that the resulting material has exactly the same distribution as dummy material computed via the trapdoor interfaces on dummy polynomials.

  Finally, for each $j \in [n]$, the simulator stores in $material[\mathsf{sid}_j]$:

$$\big(\sigma_0^\diamond[j],\ c_0^\diamond[j], \sigma^\diamond[j],\ c^\diamond[j], \Theta^\diamond[:,j],\ \Phi^\diamond[:,j], \sigma_{r_0}^\diamond[j],\ c_{r_0}^\diamond[j], \Theta_{r_0}^\diamond[:,j],\ \Phi_{r_0}^\diamond[:,j]\big),$$

Let us note that in the nominal mode $\mathcal{F}_{\mathrm{rbc}}^{\mathrm{acc}}$ ensures agreement (and so binding). If a disagreement occurs, $\mathcal{A}^{\mathrm{sim}}$ must have sent a $(\mathsf{breakAgreement}, \boldsymbol{\pi}, *)$ request, with valid proofs of misbehavior $\boldsymbol{\pi}$. In such a case, a corresponding $(\mathsf{breakAgreement}, \boldsymbol{\pi}, *)$ request is sent by $\mathcal{S}$ to $\mathcal{F}_{\mathrm{pbhacss}}^{\mathsf{AH,acc}}$

(C) Local-add operations.. Whenever a local_add operation is triggered in the simulated real world, the simulator forwards the corresponding request to the ideal functionality, and updates $material$ by applying the same $\mathbb{F}$-linear combination. Linearity of TPC and TC, together with the mixed-mode linearity of $(\mathsf{Com}, \mathsf{TCom})$ and $(\mathsf{Open}, \mathsf{TOpen})$, ensures that the resulting material is distributed exactly as in a real execution.

(D) Reconstruction and use of trapdoor collisions.. Consider a reconstruction request for a vector of session identifiers $\mathbf{sid}$, and let the ideal functionality reveal the corresponding secrets $\boldsymbol{\Xi}[1,:]$. For each involved session $\mathbf{sid}[j]$, the simulator retrieves the dummy material stored in $material[\mathbf{sid}[j]]$:

$$\big(\sigma_0^\diamond[j],\ c_0^\diamond[j],\ \sigma^\diamond[j],\ c^\diamond[j],\ \Theta^\diamond[:,j],\ \Phi^\diamond[:,j],\ \sigma_{r_0}^\diamond[j],\ c_{r_0}^\diamond[j],\ \Theta_{r_0}^\diamond[:,j],\ \Phi_{r_0}^\diamond[:,j]\big),$$

and performs the following operations.

1) **Recovering the secret-sharing polynomial.** The simulator applies the trapdoor collision algorithm of TPC to obtain the corresponding sharing polynomial:

$$\boldsymbol{\Xi}[:,j] := \mathsf{TPC.TCol}\big(\sigma^\diamond[j],\ c^\diamond[j],\ \{(\Phi^\diamond[i,j], \Theta^\diamond[i,j])\}_{i\in\mathsf{Dis}},\ \boldsymbol{\Xi}[1,j]\big).$$

   This yields the unique polynomial consistent with all dummy material and whose constant term equals the leaked secret $\boldsymbol{\Xi}[1,j]$.
2) **Recovering the base-commitment randomness.** The simulator computes the commitment randomness explaining the dummy Pedersen-like commitment $c_0^\diamond[j]$:

$$\boldsymbol{\Xi}_{r_0}[1,j] := \mathsf{TC.TCol}\big(\sigma_0^\diamond[j],\ c_0^\diamond[j],\ \boldsymbol{\Xi}[1,j]\big).$$

3) **Recovering the randomness-sharing polynomial.** Using the trapdoor collision algorithm of TPC again, the simulator reconstructs the randomness-sharing polynomial:

$$\boldsymbol{\Xi}_{r_0}[:,j] := \mathsf{TPC.TCol}\big(\sigma_{r_0}^\diamond[j],\ c_{r_0}^\diamond[j],\ \{(\Phi_{r_0}^\diamond[i,j], \Theta_{r_0}^\diamond[i,j])\}_{i\in\mathsf{Dis}},\ \boldsymbol{\Xi}_{r_0}[1,j]\big).$$

   This polynomial exactly explains the dummy randomness commitments and openings and has the required constant term $\boldsymbol{\Xi}_{r_0}[1,j]$.
4) **Regenerating honest sharing material.** With the true polynomials $(\boldsymbol{\Xi}, \boldsymbol{\Xi}_{r_0})$ recovered, the simulator invokes the honest sharing routine:

$$(\Phi^\diamond, \Theta^\diamond) := \mathsf{TPC.Share}_{\boldsymbol{\alpha}}(\boldsymbol{\Xi}), \qquad (\Phi_{r_0}^\diamond, \Theta_{r_0}^\diamond) := \mathsf{TPC.Share}_{\boldsymbol{\alpha}}(\boldsymbol{\Xi}_{r_0}),$$

   producing fresh, correctly distributed sharing material identical to that of a real honest dealer.
5) **Feeding honest simulated processes.** For each correct simulated process $\bar{p}_i^{\mathsf{sim}}$, the simulator provides the i-th reconstructed row

$$\big(\Phi^\diamond[i,:],\ \Theta^\diamond[i,:],\ \Phi_{r_0}^\diamond[i,:],\ \Theta_{r_0}^\diamond[i,:]\big),$$

   after which all simulated honest processes continue the reconstruction protocol verbatim as in a real execution.

By equivocality of TPC and TC, replacing the previously issued dummy material with fresh material explaining the corresponding polynomials is computationally indistinguishable from a real execution with an honest dealer.

Indistinguishability argument.. We now outline the indistinguishability argument between:

- the real execution of $\mathcal{P}^{\mathsf{AH,acc}}_{\text{pbhacss}}$ with adversary $\mathcal{A}$, and
- the ideal execution of $\mathcal{F}^{\mathsf{AH,acc}}_{\text{pbhacss}}$ with simulator $\mathcal{S}$.

In the honest-dealer case, the only difference between the two executions is whether the commitments and ciphertexts produced during the sharing phase encode the true polynomials (real execution) or dummy ones that are later equivocated using $\mathsf{TPC.TCol}^{\tau}$ and $\mathsf{TC.TCol}^{\tau_0}$ (ideal execution). By equivocality property, these transcripts are computationally indistinguishable.

In the corrupted-dealer case, the simulator simply reads the witnesses from the witness tape of $\mathcal{F}_{\text{zkpok}}$ and programs $\mathcal{F}^{\mathsf{AH,acc}}_{\text{pbhacss}}$ consistently. By binding of PC and knowledge soundness of $\mathcal{F}_{\text{zkpok}}$, the ideal secret is uniquely determined by the real-world behaviour of the corrupted dealer. The environment thus observes exactly the same outputs and detection events in both worlds.

Message delivery is simulated in lock-step: every message of the real execution is mirrored by $\mathcal{S}$ in the ideal execution, and vice versa. The scheduling imposed by $\mathcal{A}^{\text{sim}}$ is preserved.

Accountability.. Any violation of agreement, completeness, or binding in the real protocol is detected via the accountability mechanisms of $\mathcal{F}^{\text{acc}}_{\text{rbc}}$ and $\mathcal{F}_{\text{ack}}$ (e.g. equivocation on RBC, or inconsistent approvals). These functionalities already specify an explicit blame set G and a proof π, which $\mathcal{S}$ forwards verbatim to the environment in the ideal execution. Thus detection events occur under exactly the same conditions in both worlds.

Complexity.. The dominating communication cost stems from the ZKPoK invocation at line 14. Since the VEPC relation is a homomorphism relation between abelian groups, it admits an Ω-protocol [46], [47] whose proof size is $O(n)$ group elements (or scalars), i.e., $O(\kappa n)$ bits. For a batch size $K = \Theta(n)$, one such proof is generated per shared instance, leading to a total communication cost of $O(\kappa n^2)$. Since reliable-broadcasting message m through $\mathcal{F}^{\text{acc}}_{\text{rbc}}$ requires $O(|m|n + \kappa n^2)$ communication, reliable-broadcasting the commitments (line 16), and the certificate associated to the $\mathcal{F}_{\text{ack}}$'s acknowledgment (line 20) requires $O(\kappa n^2)$ communication.

Conclusion.. Combining the above cases, we obtain that for every real-world adversary A there exists a simulator $\mathcal{S}$ such that the ideal and real executions are computationally indistinguishable. Hence $\mathcal{P}^{\mathsf{AH,acc}}_{\text{pbhacss}}$ UC-realizes $\mathcal{F}^{\mathsf{AH,acc}}_{\text{pbhacss}}$ in the stated hybrid model. □

Appendix D. Coupled Shared Randomness

In this section, we present the accountable shared randomness $\mathcal{F}^{\text{acc}}_{\text{sr}}$ functionality, presented in Functionality 4, whose realization $\mathcal{P}^{\text{acc}}_{\text{sr}}$ is presented in Algorithm 6, and, when $(n-t-t_{acc}) \in \Omega(n)$, achieves (asymptotically) the same (amortized) per-secret communication complexity than the underlying $\mathcal{F}^{\text{acc}}_{\text{pbhacss}}$'s realization.

D.1. Co-sharing

Mask opening requires "co-sharing", i.e., sharing the same secrets through two types of secret sharing. We abstract co-sharing in Functionality 3. The corresponding realization (see Algorithm 5) is straightforward: the dealer shares the same (packed and batched) secrets through two instances of $\mathcal{F}^{\text{acc}}_{\text{pbhacss}}$, instantiated with two types of metadata, and then proves via $\mathcal{F}^{\text{1:M,acc}}_{\text{zkPoK}}$ that the two metadata values are generated from the same secret pre-image. A verifier acknowledges the co-sharing if and only if (1) it has acknowledged both underlying sharings, (2) it receives a statement from $\mathcal{F}^{\text{1:M,acc}}_{\text{zkPoK}}$, and (3) the statement matches the metadata of the two sharings.

D.1.1. Simulation-based Security Proof.

Theorem 6. *In the* $(\mathcal{F}^{\text{acc}}_{\text{pbhacss}}(\mathsf{Mt}^1), \mathcal{F}^{\text{acc}}_{\text{pbhacss}}(\mathsf{Mt}^2), \mathcal{F}^{\text{1:M,acc}}_{\text{zkPoK}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$*-hybrid model, Protocol* $\mathcal{P}^{\text{acc}}_{\text{co-sh}}$ *(Algorithm 5) UC-realizes* $\mathcal{F}^{\text{acc}}_{\text{co-sh}}$ *(Functionality 3) .*

Foreword. We consider the ideal functionality

$$\mathcal{F}^{\text{acc}}_{\text{co-sh}} \;=\; p_1^D \mid \cdots \mid p_n^D \mid \tilde{\mathcal{F}}^{\mathsf{acc}}_{\mathsf{co\text{-}sh}},$$

where $\tilde{\mathcal{F}}^{\mathsf{acc}}_{\mathsf{co\text{-}sh}}$ denotes $\mathcal{F}^{\text{acc}}_{\text{co-sh}}$ stripped of the dummy processes $p_1^D, \ldots, p_n^D$. We work in the $(\mathcal{F}^{\text{acc}}_{\text{pbhacss}}(\mathsf{Mt}^1), \mathcal{F}^{\text{acc}}_{\text{pbhacss}}(\mathsf{Mt}^2), \mathcal{F}^{\text{1:M,acc}}_{\text{zkPoK}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$-hybrid model and analyze the real protocol

$$\mathcal{P}^{\text{acc}}_{\text{co-sh}} \;=\; \bar{p}_1 \mid \cdots \mid \bar{p}_n \mid \mathcal{F}^{\text{acc}}_{\text{pbhacss}}(\mathsf{Mt}^1) \mid \mathcal{F}^{\text{acc}}_{\text{pbhacss}}(\mathsf{Mt}^2) \mid \mathcal{F}^{\text{1:M,acc}}_{\text{zkPoK}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2)),$$

whose pseudocode is given in Algorithm 5.

Functionality 3 $\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1), \mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)\rangle$

Subroutines:
$\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^x)$ for each $x \in \{1,2\}$, parametrized with P_x, K_x for packing and batching size, with $P_x K_x = B$ ▷ See Functionality 1

Interface:
request $(\mathsf{co_share}, \mathsf{sid}, \mathbf{S})$: Co-sharing secrets $\mathbf{S}$ under $\mathsf{sid} \in \mathsf{SSID}_{co}$.
notification $(\mathsf{co_share_ack}, \mathsf{sid})$: Acknowledging the co-sharing under $\mathsf{sid} \in \mathsf{SSID}_{co}$.
all other requests and notifications as defined by $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}$.
$(\mathsf{Sh.share}, \mathsf{sid} \circ x, *)$ with $x \in \{1,2\}$ is ignored by $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^x)$ if $\mathsf{sid} \in \mathsf{SSID}_{co}$

Functionality:
Upon receiving $(\mathsf{co\text{-}share}, \mathsf{sid}, \mathbf{S})$ from any party D: ▷ For allowed SIDs only
$\mathbf{S}' \leftarrow \mathsf{lift}_{(P_2,K_2)}(\mathbf{S})$, i.e.
$\forall (i_2, j_2) \in [P_2] \times [K_2]$: $\mathbf{S}'[i_2, j_2] := \mathbf{S}[i_1, j_1]$, with $(i_1, j_1) = \mu^{-1}_{(P_1,K_1)}(\mu_{(P_2,K_2)}(i_2, j_2))$, where $\mu_{(P_x,K_x)} : (i_x, j_x) \in (P_x, K_x) \mapsto (K_x(i_x - 1) + j_x) \in [B]$ for each $x \in \{1,2\}$
  Send $(\mathsf{Sh.share}, \mathsf{sid} \circ 1, \mathbf{S})$ to $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1)$ on behalf of D.
  Send $(\mathsf{Sh.share}, \mathsf{sid} \circ 2, \mathbf{S}')$ to $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)$ on behalf of D
  Upon receiving $(\mathsf{Sh.ack}, \mathsf{sid} \circ x, m^x)$ from $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^x)$ $\forall x \in \{1,2\}$, for $p \in \Psi$:
    Send $(\mathsf{co_share_ack}, \mathsf{sid})$ to $p \in \Psi$ as a delayed output

Upon receiving any notification from $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1)$ or $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)$:
  Forward the notification verbatim to the corresponding recipients.

Upon receiving any request intended for $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1)$ or $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)$:
  Forward the request verbatim to the corresponding subroutine.

Algorithm 5 $\mathcal{P}_{\text{co-sh}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1), \mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)\rangle$

Use.
  $\forall x \in \{1,2\}$: $\mathcal{F}_{\text{bphacss}}^{\text{acc}}(\mathsf{Mt}^x)$ parametrized with P_x, K_x s.t $P_x K_x = B$, and where $(\mathsf{Sh.share}, \mathsf{sid} \circ x, *)$ sent by $p \in \Psi$ is ignored by $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^x)$ if $\mathsf{sid} \in \mathsf{SSID}_{co}$
  $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}(\mathsf{Mt}^1, \mathsf{Mt}^2))$, where
  $\mathcal{R}(\mathsf{Mt}^1, \mathsf{Mt}^2) = \{\Big((\mathbf{S}, \mathbf{r}^1, \mathbf{r}^2), (\mathsf{Mt}^1(\mathbf{S}, \mathbf{r}^1), \mathsf{Mt}^2(\mathsf{lift}_{(P_2,K_2)}(\mathbf{S}), \mathbf{r}^2))\Big)\}$.

Local state.
  $\mathsf{Map}(\mathsf{SID} \to \mathsf{String}^4)$ $meta[][]$ ▷ $[\perp, \perp, \perp, \perp]$

Co-sharing (dealer)
upon request $(\mathsf{co\text{-}share}, \mathsf{sid}, \mathbf{S})$ with $\mathsf{sid} \in \mathsf{SSID}_{\mathsf{co}}$:
  send $(\mathsf{Sh.share}, \mathsf{sid} \circ 1, \mathbf{S})$ to $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1)$
  send $(\mathsf{Sh.share}, \mathsf{sid} \circ 2, \mathsf{lift}_{(P_2,K_2)}(\mathbf{S}))$ to $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)$
  upon receiving $(\mathsf{Sh.receive}, \mathsf{sid} \circ x, \mathbf{r}^x, m^x)$ from $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^x)$ $\forall x \in \{0,1\}$:
    $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}(\mathsf{Mt}^1, \mathsf{Mt}^2))$.$(\mathsf{prove}, \mathsf{sid}, (\mathbf{S}, \mathbf{r}^1, \mathbf{r}^2), (m^1, m^2))$

Co-sharing (verifiers)
upon notification $(\mathsf{Sh.ack}, \mathsf{sid} \circ 1, m^1)$ from $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1)$:
  $meta[\mathsf{sid}][1] \leftarrow m^1$
upon notification $(\mathsf{Sh.ack}, \mathsf{sid} \circ 2, m^2)$ from $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)$:
  $meta[\mathsf{sid}][2] \leftarrow m^2$
upon $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}(\mathsf{Mt}^1, \mathsf{Mt}^2))$.$(\mathsf{statement}, \mathsf{sid}, (\tilde{m}^1, \tilde{m}^2))$:
  $(meta[\mathsf{sid}][3], meta[\mathsf{sid}][4]) \leftarrow (\tilde{m}^1, \tilde{m}^2)$
upon $(meta[\mathsf{sid}][1], meta[\mathsf{sid}][2]) = (meta[\mathsf{sid}][3], meta[\mathsf{sid}][4])$:
  trigger $(\mathsf{co\text{-}share_ack}, \mathsf{sid})$.

Forwarding
upon any request $(\mathsf{Req}, \mathsf{sid}, \ldots)$ with $\mathsf{sid} \notin \mathsf{SSID}^{\mathsf{co}}$:
  forward verbatim to the corresponding subroutine.
upon any notification $(\mathsf{Notif}, \mathsf{sid}^i, \ldots)$ from $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^i)$ for $i \in \{1,2\}$:
  forward verbatim (replacing sid^i by sid when appropriate).

Let $\mathcal{A}$ be an arbitrary real-world adversary for $\mathcal{P}_{\text{pbhacss}}^{\text{acc}}$. We construct a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\text{co-sh}}^{\text{acc}}$ and the environment $\mathcal{E}$ such that $\mathcal{E}$ cannot distinguish the ideal execution from the real one. The simulator $\mathcal{S}$ internally runs a copy of $\mathcal{A}$, denoted $\mathcal{A}^{\text{sim}}$. For each process $\bar{p}_k$, $\mathcal{S}$ simulates a copy $\bar{p}_k^{\mathsf{sim}}$ (honest dealers are obviously not fully simulated due to privacy constraints). Whenever $\bar{p}_k^{\mathsf{sim}}$ is corrupted by $\mathcal{A}^{\text{sim}}$, $\mathcal{S}$ corrupts the corresponding dummy process p_k^D in the ideal world. Additionally, $\mathcal{S}$ simulates (pseudo) copies $\mathcal{F}_{\text{pbhacss}}^{\text{acc,sim}}(\mathsf{Mt}^1)$, $\mathcal{F}_{\text{pbhacss}}^{\text{acc,sim}}(\mathsf{Mt}^2)$, and $\mathcal{F}_{\text{zkpok}}^{1:\text{M},\text{acc,sim}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$ of $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^1)$, $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{Mt}^2)$, and $\mathcal{F}_{\text{zkpok}}^{1:\text{M},\text{acc}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$, respectively, with the following modification: for session IDs associated with honest provers, $\mathcal{S}$ writes the statement directly on the output tape of $\mathcal{F}_{\text{zkpok}}^{1:\text{M},\text{acc,sim}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$, while $\mathcal{F}_{\text{zkpok}}^{1:\text{M},\text{acc,sim}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$ never receives the corresponding witness on its input tape. Similarly, for each $x \in \{1,2\}$, $\mathcal{S}$ writes the metadata directly on the output tape of $\mathcal{F}_{\text{pbhacss}}^{\text{acc,sim}}(\mathsf{Mt}^x)$, while $\mathcal{F}_{\text{pbhacss}}^{\text{acc,sim}}(\mathsf{Mt}^x)$ never receives the corresponding

secrets on its input tape.

Finally, whenever $\bar{p}_i^{\mathrm{sim}}$ triggers $(\mathsf{detection}, \mathsf{sid}_*, G_i, \pi_i)$, the simulator orders $\mathcal{F}_{\mathrm{btg}}^{\mathrm{acc}}$ to forward the corresponding $(\mathsf{detection}, \mathsf{sid}, G_i, \pi_i)$ notification to p_i.

Messages from $\mathcal{E}$ to $\mathcal{S}$ (resp., a corrupted dummy process p_k^D) are forwarded to $\mathcal{A}^{\mathrm{sim}}$ (resp., $\bar{p}_k^{\mathrm{sim}}$), and messages from $\mathcal{A}^{\mathrm{sim}}$ (resp., $\bar{p}_k^{\mathrm{sim}}$) to $\mathcal{E}$ are forwarded back verbatim by $\mathcal{S}$ (resp., p_k^D).

Simulating Co-Sharing.. Consider a sharing request $\big(\mathsf{Sh.share}, \mathsf{sid}, \mathbf{S}\big)$ issued by a dealer D.

Case 1: D **is honest.** In the ideal world, $\mathcal{F}_{\text{co-sh}}^{\mathrm{acc}}$ receives the sharing request and, for each $x \in \{1,2\}$, forwards it to $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc}}(\mathsf{Mt}^x)$ that stores the secrets, samples randomness $\boldsymbol{r}_x$ and generates metadata $m^x = \mathsf{Mt}^x(\boldsymbol{S}^x, \boldsymbol{r}^x)$, with $(\boldsymbol{S}^1, \boldsymbol{S}^2) = (\boldsymbol{S}, \mathsf{lift}_{(P_2,K_2)}(\boldsymbol{S}))$, and and leaks it to the ideal adversary. Upon such a notification, $\mathcal{S}$ writes m^1, m^2, and (m^1, m^2), directly on the output tapes of $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc,sim}}(\mathsf{Mt}^1)$, $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc,sim}}(\mathsf{Mt}^2)$, and $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc,sim}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$, respectively.

Case 2: D **is corrupted and no** BreakAgreement **command has been registered.** To be acknowledged, any co-sharing performed in the simulated real world must have been preceded by a corresponding notification from $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc,sim}}$, itself preceded by the dealer writing (1) witness–statement pair

$$\big((\boldsymbol{S}, \boldsymbol{r}^1, \boldsymbol{r}^2), (\mathsf{Mt}^1(\boldsymbol{S}, \boldsymbol{r}^1), \mathsf{Mt}^2(\mathsf{lift}_{(P_2,K_2)}(\boldsymbol{S}), \boldsymbol{r}^2)\big)$$

on the corresponding input tape of $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc,sim}}(R(\mathsf{Mt}^1, \mathsf{Mt}^2))$, and (2) $\boldsymbol{S}^x$ on the corresponding input tape of $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc,sim}}(\mathsf{Mt}^x)$ for each $x \in \{1,2\}$.

Thus, the simulator can simply read $\boldsymbol{S}$ and feed $\mathcal{F}_{\text{co-sh}}^{\mathrm{acc}}$ accordingly. Consistency is maintained as long as for any $x \in \{1,2\}$, $\mathcal{A}$ cannot break the binding property of Mt^x, i.e. computing $(\boldsymbol{S}, \boldsymbol{S}', \boldsymbol{r}, \boldsymbol{r}')$ such that $\mathsf{Mt}^x(\boldsymbol{S}, \boldsymbol{r}) = \mathsf{Mt}^x(\boldsymbol{S}', \boldsymbol{r}')$.

Case 3: D **is corrupted and** BreakAgreement **has been registered.** For each session ID, the simulator $\mathcal{S}$ maintains 3 mappings $\underline{\mathbf{S}}_1, \underline{\mathbf{S}}_2, \underline{\mathbf{S}}_{zk} : [n] \to \mathsf{Mat}_{(P,K)}$. These correspond to the disagreements induced by the two instances of $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc}}$ and by $\mathcal{F}_{\mathrm{zkPoK}}^{\mathrm{1:M,acc}}$, respectively. If $\mathcal{A}^{\mathrm{sim}}$ sends $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \underline{\mathbf{S}}_x)$ to $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc,sim}}\langle \mathsf{Mt}^x \rangle$ or $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \underline{\mathbf{S}}_{zk})$ to $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc,sim}}$, then $\mathcal{S}$ sends to $\mathcal{F}_{\text{co-sh}}^{\mathrm{acc}}$ both the commands $(\mathsf{BreakCompleteness}, \mathsf{sid})$ and $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \underline{\mathbf{S}}_x/\underline{\mathbf{S}}_{zk})$, and updates $\underline{\mathbf{S}}_x$ or $\underline{\mathbf{S}}_{zk}$ accordingly. Whenever $\underline{\mathbf{S}}[i] \neq \underline{\mathbf{S}}'[i]$ for $i \in [n]$ and $\underline{\mathbf{S}}, \underline{\mathbf{S}}' \in \{\underline{\mathbf{S}}_1, \underline{\mathbf{S}}_2, \underline{\mathbf{S}}_{zk}\}$, the simulator $\mathcal{S}$ indefinitely delays the sharing acknowledgement notification to process p_i. Later, when a simulated process $\bar{p}_i^{\mathrm{sim}}$ triggers $(\mathsf{detection}, \mathsf{sid}_*, G_i, \pi_i)$, $\mathcal{S}$ triggers the corresponding delayed output $(\mathsf{detection}, \mathsf{sid}, G_i, \pi_i)$ to p_i via $\mathcal{F}_{\text{co-sh}}^{\mathrm{acc}}$.

Processes interact exclusively via $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc}}$ $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc}}$. Moreover, the soundness of $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc}}$ combined with the binding property of Mt^1 and Mt^2 implies that an acknowledged co-sharing implies the sharing of the same secrets for the two underlying secret-sharing subroutines. This means that a breach in the simulation can only be caused by a safety violation in $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc}}$ or $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc}}$, which necessarily yields a detection notification with an associated externally verifiable proof $\boldsymbol{\pi}$. In such a case, $\mathcal{S}$ triggers the corresponding BreakAgreement request with $\boldsymbol{\pi}$ attached, matching the real protocol's behavior.

Indistinguishability Argument.. We have thus constructed a forward simulation from the real world to the ideal world, which makes them indistinguishable.

Message delivery and fairness.. Liveness (in particular, completeness and obligation) is ensured by the liveness properties of $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc}}$ and $\mathcal{F}_{\mathrm{zkpok}}^{\mathrm{1:M,acc}}$.

This concludes the proof that $\mathcal{P}_{\text{co-sh}}^{\mathrm{acc}}$ UC-realizes $\mathcal{F}_{\text{co-sh}}^{\mathrm{acc}}$ in the stated hybrid model.

D.2. Coupled shared randomness

The functionality $\mathcal{F}_{\text{co-sr}}^{\mathrm{acc}}\langle \mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc},1}, \mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc},2} \rangle$ (see Functionality 4) generates randomness that is consistently shared across two accountable packed–batched PBHACSS instances, potentially instantiated with different packing and batching parameters but satisfying $P_1K_1 = P_2K_2$. Parties initiate generation via $(\mathsf{Gen.enable}, \mathsf{sid})$. Once at least $n - t$ correct parties have done so, the functionality samples, for each batch index k and packing index q, a vector of n field elements, anticipating possible disagreement. In the nominal case, all parties receive the same value, which is installed consistently into both $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc},1}$ and $\mathcal{F}_{\mathrm{pbhacss}}^{\mathrm{acc},2}$ according to the fixed index mapping η. The functionality then acknowledges completion by issuing $(\mathsf{Gen.receive}, \mathsf{sid})$.

In the degraded agreement regime ($f > t$), the adversary may trigger the command $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mathsf{map})$, in which case the functionality exposes the session, installs per-party inconsistent randomness according to map, and outputs detection events identifying a blame set of size at least d, provided the judge validates the accompanying proofs. In the degraded liveness regime, upon $(\mathsf{BreakLiveness}, \mathsf{sid})$, all pending notifications except detection may be delayed indefinitely.

Explicit metadata biasing mechanism. The following metadata bias machinery is omitted for the sake of presentation:

1) **Initial sampling.** Upon sampling a secret $s \leftarrow_{\$} \mathbb{F}$, the functionality samples fresh randomness r_m and computes

$$m \;=\; \mathsf{Mt}^x(s; r_m),$$

which is revealed to the (ideal) adversary.

Functionality 4 $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}(\mathsf{Mt}^1),\ \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}(\mathsf{Mt}^2)\rangle$ – Co-Shared Randomness

Use:
For each $x \in \{1,2\}$: $\mathcal{F}_{\text{pbhacss}}^{\text{acc},x}$ parametrized with $(P_x, K_x) \in \mathbb{N}^2$; $P_1K_1 = P_2K_2$;
For each $x \in \{1,2\}$, $\mu_x : (q_x, k_x) \in [P_x] \times [K_x] \mapsto K_x(q_x - 1) + k_x$; $\eta = \mu_2^{-1} \circ \mu_1$

Variables:
$\mathsf{Map}(\mathsf{SID} \to \mathsf{Set}(\mathsf{Process}))$ $I[]$ (init $\emptyset$): generation initiators per top-level session sid
$\mathsf{Map}(\mathsf{SID} \to \mathsf{Boolean})$ $exposed[]$ (init $false$)
$\mathsf{Map}(\mathsf{SID} \times [\mathsf{P}] \times \mathsf{PID} \to \mathbb{F})$ $rand[][]$ (init $\perp$)

Events / Interface:
request $(\mathsf{Gen.enable}, \mathsf{sid})$: Initiating randomness generation
notification $(\mathsf{Gen.receive}, \mathsf{sid})$: Acknowledging randomness generation
notification $(\mathsf{detection}, \mathsf{sid}, G, \pi^{pom})$: Getting a proof π^{pom} incriminating culprits G

Functionality:
Upon receiving $(\mathsf{Gen.enable}, \mathsf{sid})$ from p_i:
  $I[\mathsf{sid}] \leftarrow I[\mathsf{sid}] \cup \{p_i\}$; forward $(\mathsf{Gen.enable}, \mathsf{sid}, p_i)$ to Sim.

Upon $|I[\mathsf{sid}] \cap \mathsf{Correct}| \geq n - t$ with $exposed[\mathsf{sid}] = false$:
  $\forall (k,q) \in [K_1] \times [P_1] : rand[\mathsf{sid} \circ k][q] \leftarrow_{\$} \mathbb{F}^n$ ▷ anticipates disagreement
  For each $(i,k,q) \in [n] \times [K] \times [P]$:
    $\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}.secret[\mathsf{sid} \circ k][q][i] \leftarrow rand[\mathsf{sid} \circ k][q][1]$, ▷ Agreement
    $\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}.secret[\mathsf{sid} \circ k'][q'][i] \leftarrow rand[\mathsf{sid} \circ k][q][1]$, ▷ Agreement
    where $(q', k') = \eta(q, k)$
  Notify Sim and sends $(\mathsf{Gen.receive}, \mathsf{sid})$ to each p_i as a delayed output.

Upon receiving $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mathsf{map} : [n] \to [n])$ with $f > t$:
  $exposed[\mathsf{sid}] \leftarrow true$.
  send $(\mathsf{detection}, \mathsf{sid}, G_i, \boldsymbol{\pi}[i])$ to each p_i as delayed output, with $|G_i| \geq d$, s.t.
  the judge returns the (fair) verdict $\mathsf{dis}(G_i)$ on evidence $\boldsymbol{\pi}[i]$.
  For each $(i,k,q) \in [n] \times [K] \times [P]$:
    $\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}.secret[\mathsf{sid} \circ k][q][i] \leftarrow rand[\mathsf{sid} \circ k][q][\mathsf{map}(i)]$, ▷ Disagreement
    $\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}.secret[\mathsf{sid} \circ k'][q'][i] \leftarrow rand[\mathsf{sid} \circ k][q][\mathsf{map}(i)]$. ▷ Disagreement
    where $(q', k') = \eta(q, k)$

Upon receiving $(\mathsf{BreakLiveness}, \mathsf{sid})$ with $f > t$: ▷ Liveness violation
  Any scheduled notification (expect detection) can be indefinitely delayed

2) **Adversarial bias.** The adversary may choose an offset

$$m' \ = \ \mathsf{Mt}^x(s'; r'_m),$$

for arbitrary $s' \in \mathbb{F}$ and randomness r'_m, thereby biasing the metadata homomorphically.

3) **Final installation.** The functionality sets

$$s'' \ = \ s + s', \qquad r''_m \ = \ r_m + r'_m,$$

and publishes the effective metadata

$$m'' \ = \ m \oplus m' \ = \ \mathsf{Mt}^x(s''; r''_m),$$

which is consistent with the doubly additive homomorphism of Mt^x.

Realization.. The protocol $\mathcal{P}_{\text{co-sr}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ (see Algorithm 6) realizes $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$ in the $\big(\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle, \mathcal{F}_{\text{idx_acs}}^{\text{acc}}\big)$-hybrid model. Each party locally samples a matrix of random values and co-shares it using the co-sharing functionality, thereby installing consistent sharings in both PBHACSS instances. The parties then invoke $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$ to agree on a core set Q of size $n - t$. From the sharings of the parties in Q, they extract $(n - t - t_{acc})$ batches of shared randomness by applying a fixed hyperinvertible matrix $\mathbf{V}$ via local linear combinations in each PBHACSS instance. The resulting values are consistently indexed across the two instances according to the prescribed mapping and constitute the output of the shared-randomness generation. Finally, the protocol triggers $(\mathsf{Gen.receive}, \mathsf{sid})$.

Any detection event produced by either the co-sharing functionality or $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$ is forwarded verbatim to the output interface, ensuring that violations of agreement are accompanied by publicly verifiable evidence.

Theorem 7 (Shared Randomness). *Protocol* $\mathcal{P}_{\text{co-sr}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ *Algorithm 6 realizes* $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ *of Functionality 4 in the* $\big(\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle, \mathcal{F}_{\text{idx_acs}}^{\text{acc}}\big)$*-hybrid model.*

Let $\mathcal{A}$ be an arbitrary real-world adversary for $\mathcal{P}_{\text{sr}}^{\text{acc}}$. We construct a simulator $\mathcal{S}$ interacting with the ideal functionality $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ and running a copy $\mathcal{A}^{\text{sim}} \mid \mathcal{P}_{\text{sr}}^{\text{acc,sim}}$ of the real protocol internally. $\mathcal{S}$ has access to the same hybrid subroutines as the real protocol, namely $\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ and $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$, and simulates all honest parties' code of Algorithm 6 for $\mathcal{A}$.

Algorithm 6 $\mathcal{P}_{\text{sr}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhass}}^{\text{acc},1}, \mathcal{F}_{\text{pbhass}}^{\text{acc},2} \rangle$ – $\mathcal{F}_{\text{sr}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhass}}^{\text{acc},1}, \mathcal{F}_{\text{pbhass}}^{\text{acc},2} \rangle$'s realization

Parameters:
base field $\mathbb{F}$; $n, t, t_{acc}, P_1, K_1 \in \mathbb{N}$; $P_2 = 1, K_2 = P_1 K_1$
fixed hyperinvertible (e.g. Vandermonde) matrix $\mathbf{V} \in \mathsf{Mat}_{\left((n-t),(n-t-t_{acc})\right)}(\mathbb{F})$

Use:
$\mathcal{F}_{\text{co-sharing}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhass}}^{\text{acc},1}, \mathcal{F}_{\text{pbhass}}^{\text{acc},2} \rangle$ and $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$.

Sharing phase
upon request $(\mathsf{Gen.enable}, \mathsf{sid})$ by p_i:
  sample $\mathbf{S} \leftarrow_{\$} \mathsf{Mat}_{(P_1,K_1)}(\mathbf{S})$
  $\mathcal{F}_{\text{co-sharing}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhass}}^{\text{acc},1}, \mathcal{F}_{\text{pbhass}}^{\text{acc},2} \rangle.(\mathsf{co_share}, \mathsf{sid} \circ i, \mathbf{S})$

upon $\mathcal{F}_{\text{co-sharing}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhass}}^{\text{acc},1}, \mathcal{F}_{\text{pbhass}}^{\text{acc},2} \rangle.(\mathsf{co_share_ack}, \mathsf{sid} \circ j)$
  send $(\mathsf{acs_validate}, \mathsf{sid}, p_j)$ to $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$.

Agreement on a core set and random extraction
upon $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}.(\mathsf{acs_decide}, \mathsf{sid}, Q = \{i_1, \ldots, i_{n-t}\})$:
  for each $(k, j) \in [K_1] \times [n - t - t_{acc}]$:
    $\mathcal{F}_{\text{pbhass}}^{\text{acc},1}.(\mathsf{local_add},\ \mathsf{sid}_{(k,j)},\ [\mathsf{sid}_{(i_1,k)}, \ldots, \mathsf{sid}_{(i_{n-t},k)}],\ \mathbf{V}[:, j])$.
    for each $q \in [P_1]$:
      $k' := k + K_1(q - 1)$
      $\mathcal{F}_{\text{pbhass}}^{\text{acc},2}.(\mathsf{local_add},\ \mathsf{sid}_{(k',j)},\ [\mathsf{sid}_{(i_1,k')}, \ldots, \mathsf{sid}_{(i_{n-t},k')}],\ \mathbf{V}[:, j])$.
  trigger notification $(\mathsf{Gen.receive}, \mathsf{sid})$.

Accountability
upon notification $(\mathsf{detection}, \mathsf{sid}_*, G, \pi)$ from $\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhass}}^{\text{acc},1}, \mathcal{F}_{\text{pbhass}}^{\text{acc},2} \rangle$ or $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$:
  trigger notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$.

External interface simulation.

- When $\mathcal{S}$ receives $(\mathsf{Gen.enable}, \mathsf{sid}, p_i)$ from $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$, it forwards $(\mathsf{Gen.enable}, \mathsf{sid})$ to the corresponding simulated process $\bar{p}_i^{\text{sim}}$ in $\mathcal{P}_{\text{sr}}^{\text{acc,sim}}$. Conversely, when $\bar{p}_i^{\text{sim}}$ triggers $(\mathsf{Gen.receive}, \mathsf{sid})$, $\mathcal{S}$ triggers the delayed output $(\mathsf{Gen.receive}, \mathsf{sid})$ to p_i via $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$.
- When $\mathcal{A}^{\text{sim}}$ sends $(\mathsf{BreakLiveness}, \mathsf{sid})$, $\mathcal{S}$ forwards $(\mathsf{BreakLiveness}, \mathsf{sid})$ to $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$.
- The simulator $\mathcal{S}$ maintains two mappings $\mu_{acs}, \mu_{co} : [n] \to [n]$, both initially defined as the constant map $i \mapsto 1$. These correspond to the partitionings induced by $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$ and by $\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2} \rangle$, respectively.
  If $\mathcal{A}^{\text{sim}}$ sends $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mu_{co})$ to $\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2} \rangle$ or sends $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \boldsymbol{Q}, \mu_{acs})$ to $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$, then $\mathcal{S}$ forwards to $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$ the command $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \tilde{\mu})$, where $\tilde{\mu}$ is any mapping such that
  $$\tilde{\mu}(i) = \tilde{\mu}(j) \iff (\mu_{acs}(i), \mu_{co}(i)) = (\mu_{acs}(j), \mu_{co}(j)).$$
  That is, the partition induced by $\tilde{\mu}$ is the common refinement of the partitions induced by μ_{acs} and μ_{co}.
- Later, when a simulated process $\bar{p}_i^{\text{sim}}$ triggers $(\mathsf{detection}, \mathsf{sid}_*, G_i, \pi_i)$, $\mathcal{S}$ triggers the corresponding delayed output $(\mathsf{detection}, \mathsf{sid}, G_i, \pi_i)$ to p_i via $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$.

Case analysis.. We distinguish the nominal and degraded regimes.

Case 1: Nominal regime ($f \leq t$).. In this regime, $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$ never executes BreakAgreement nor BreakLiveness and installs globally consistent randomness.

In the simulated real execution, $\mathcal{S}$ runs $\mathcal{P}_{\text{sr}}^{\text{acc}}$ honestly for all honest parties: each party samples a random matrix $\mathbf{S}$, co-shares it using $\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2} \rangle$, and the parties invoke $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$ to agree on a core set Q of size $n - t$. The protocol then applies the fixed hyperinvertible matrix $\mathbf{V}$ via local linear combinations in both PBHACSS instances, yielding $(n - t - t_{acc})$ batches of shared randomness consistently indexed via η.

Since co-sharing enforces equality across the two PBHACSS instances and $\mathbf{V}$ is full rank, the resulting values are uniform, independent, and identical for all honest parties, exactly as specified by $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$. The simulator forwards $(\mathsf{Gen.receive}, \mathsf{sid})$ at the corresponding point. The environment's view is therefore identical.

Case 2: Degraded agreement with accountability ($f > t$ and BreakAgreement).. Disagreement may arise either during co-sharing or during the ACS/extraction phase, inducing independent partitionings μ_{acs} and μ_{co}. The simulator forwards to $\mathcal{F}_{\text{co-sr}}^{\text{acc}}$ the common refinement $\tilde{\mu}$ of these partitionings, thereby preserving exactly the joint disagreement structure of the real execution. By construction, any pair of parties that disagree in either subroutine are separated under $\tilde{\mu}$, while parties that agree in both remain grouped.

Finally, whenever $\bar{p}_i^{\text{sim}}$ triggers $(\mathsf{detection}, \mathsf{sid}_*, G_i, \pi_i)$, the simulator orders $\mathcal{F}_{\text{btg}}^{\text{acc}}$ to forward the corresponding $(\mathsf{detection}, \mathsf{sid}, G_i, \pi_i)$ notification to p_i. Therefore, every real-world violation of agreement that is detectable is matched by an ideal detection event with the same culprit set, and the environment's view of disagreement and accountability is indistinguishable from that of the real execution.

Fairness.. Liveness is guaranteed by the liveness of the underlying functionalities $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$ and $\mathcal{F}_{\text{co-sh}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$, and therefore holds as long as $\mathcal{A}$ does not issue any valid request $(\mathsf{BreakCompleteness}, \mathsf{sid})$. If such a request is issued, $\mathcal{S}$ forwards the command $(\mathsf{BreakCompleteness}, \mathsf{sid})$ to $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$, thereby justifying the loss of liveness in the ideal execution.

Conclusion.. We have thus constructed a (perfect) forward simulation from the real world to the ideal world, which makes them perfectly indistinguishable. The simulation preserves fair executions. Therefore, protocol $\mathcal{P}_{\text{co-sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ realizes $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ in the $(\mathcal{F}_{\text{co-sh}}^{\text{acc}}, \mathcal{F}_{\text{idx_acs}}^{\text{acc}})$-hybrid model.

Corollary 8. *$\mathcal{P}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}\rangle$ UC-realizes $\mathcal{F}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}\rangle$ in the $(\mathcal{F}_{\text{pbhacss}}, \mathcal{F}_{\text{idx_acs}}^{\text{acc}})$-hybrid model with $\mathcal{P}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}\rangle$ and $\mathcal{F}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}\rangle$ defined in the obvious manner.*

Appendix E. Beaver Triple Migration

In this section, we show how Beaver triples generated into an additively homomorphic secret sharing functionality are migrated into another one. The migration, realized by mask opening, is presented in Algorithm 7.

Functionality 5 $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}\rangle$ – Accountable Beaver Triple Generation

Parameters:
$\mathcal{F}_{\text{pbhacss}}^{\text{acc}}$ parametrized with K (batching), $P = 1$ (packing), ℓ_r (thresholds), $\mathbb{F}$ (field)
Random Extraction parameter $E = (n - t - t_{acc}) \in \mathbb{N}$; $B = KE$

Variables:
$\mathsf{Map}(\mathsf{SID} \times \mathsf{PID} \to \mathbb{F})$ $secret[][]$ points to $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}.secret[][]$ ▷ $(P = 1)$
$\mathsf{Map}(\mathsf{SID} \to \mathsf{Set}(\mathsf{PID}))$ $I[]$ (init $\emptyset$): Initiators per session

Events / Interface:
request $(\mathsf{Gen.enable}, \mathsf{sid})$: Enabling triple generation
notification $(\mathsf{Gen.ack}, \mathsf{sid})$: Acknowledging triple generation
notification $(\mathsf{detect}, \mathsf{sid}, G, \pi^{pom})$: Getting a proof π^{pom} incriminating a set G of culprits

Functionality:
Upon receiving $(\mathsf{Gen.enable}, \mathsf{sid})$ from process p_i:
  $I[\mathsf{sid}] \leftarrow I[\mathsf{sid}] \cup \{i\}$; forward $(\mathsf{Gen.enable}, \mathsf{sid}, p_i)$ to Sim.

Upon $|I[\mathsf{sid}] \cap \mathsf{Correct}| = \ell_r + 1$:
  $\forall k \in [B]$: $(a_k, b_k) \leftarrow_{\$} \mathbb{F}^2$, $c_k = a_k \otimes_{\mathbb{F}} b_k$ ▷ Beaver triple
  $\forall (d, k, i) \in \{a, b, c\} \times [B] \times [n]$: $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}.secret[\mathsf{sid}_k^d][i] \leftarrow d_k$ ▷ Agreement
  send $(\mathsf{Gen.ack}, \mathsf{sid})$ to each process as a delayed output.

Upon receiving $(\mathsf{BreakCompleteness}, \mathsf{sid})$ from Sim with $f > t$:
  any notification (except detect) can be indefinitely delayed.

Upon receiving $(\mathsf{BreakAgreement}, \mathsf{sid}, \mathsf{map} : [n] \to [n], \boldsymbol{\pi})$ with $f \in (t, t_{acc}]$:
  If, $\forall i \in [n]$, the judge returns $\mathsf{dis}(G_i)$ on top of $\boldsymbol{\pi}[i]$, with $|G_i| \geq d$:
    send $(\mathsf{detection}, \mathsf{sid}, G_i, \boldsymbol{\pi}[i])$ to each p_i as a delayed output.
    $\forall (k, i) \in [B] \times [n]$: $(a_k^i, b_k^i) \leftarrow_{\$} \mathbb{F}^2$, $c_k^i = a_k^i \otimes_{\mathbb{F}} b_k^i$ ▷ Randomness preserved
    $\forall (d, k, i) \in \{a, b, c\} \times [B] \times [n]$: $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}$: $secret[\mathsf{sid}_k^d][i] \leftarrow d_k^{\mathsf{map}(i)}$

Functionality 6 $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{bhacss}}^{\text{acc}}\rangle$ – Helper for batched multiplication

Upon $(\mathsf{mult}, \mathbf{sid}_z, \mathbf{sid}_x, \mathbf{sid}_y, \mathbf{sid}_a, \mathbf{sid}_b, \mathbf{sid}_c)$: ▷ $\mathbf{sid}_w$ corresponds to value w
1: Assert $|\mathbf{sid}_z| = |\mathbf{sid}_x| = |\mathbf{sid}_y| = |\mathbf{sid}_a| = |\mathbf{sid}_b| = |\mathbf{sid}_c|$. ▷ Batch
2: $\forall k \in [|\mathbf{sid}_z|]$:
3: Invoke $\mathcal{F}_{\text{bhacss}}^{\text{acc}}.\mathsf{local_add}(\mathbf{sid}_{\tilde{A}}[k], [\mathbf{sid}_x[k], \mathbf{sid}_a], [1, -1])$ ▷ $\tilde{A} = (x - a)$
4: Invoke $\mathcal{F}_{\text{bhacss}}^{\text{acc}}.\mathsf{local_add}(\mathbf{sid}_{\tilde{B}}[k], [\mathbf{sid}_y[k], \mathbf{sid}_b], [1, -1])$ ▷ $\tilde{B} = (y - b)$
5: Invoke $\mathcal{F}_{\text{bhacss}}^{\text{acc}}.\mathsf{Rec.enable}(\mathbf{sid}_{\tilde{A}} :: \mathbf{sid}_{\tilde{B}})$ ▷ Batched Reconstruction
6: Wait until, $\forall (\tilde{D}, k) \in \{\tilde{A}, \tilde{B}\} \times [|\mathbf{sid}_z|]$, $\mathcal{F}_{\text{bhacss}}^{\text{acc}}.\mathsf{Rec.reconstruct}(\mathbf{sid}_{\tilde{D}}[k], s_{\tilde{D}}^k)$
7: $\forall k \in [|\mathbf{sid}_z|]$:
8: $s_{\tilde{A}\tilde{B}}^k \leftarrow (s_{\tilde{A}}^k \odot_{\mathbb{F}} s_{\tilde{B}}^k)$
9: $\mathcal{F}_{\text{bhacss}}^{\text{acc}}.\mathsf{local_add}(\mathbf{sid}_z[k], [\mathbf{sid}_a[k], \mathbf{sid}_b[k], \mathbf{sid}_c[k], \mathsf{sid}_{1_{\mathbb{F}_p}}, [s_{\tilde{B}}^k, s_{\tilde{A}}^k, 1, s_{\tilde{A}\tilde{B}}^k])$ ▷ $[xy] \leftarrow (x - a)(y - b) \oplus [b](x - a) \oplus [a](y - b) \oplus [c]$ for each k. $\mathsf{sid}_{1_{\mathbb{F}_p}}$ stores publicly known constant $1_{\mathbb{F}_p}$.

The ideal functionality $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc}}\rangle$ has a very small external interface: upon sufficiently many Gen.enable requests, it installs B Beaver triples $(a_k, b_k, c_k = a_k b_k)$ into the underlying $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}$ state (agreement), and triggers $(\mathsf{Gen.ack}, \mathsf{sid})$; additionally, it supports the usual two adversarial "break" hooks: BreakCompleteness (delay) and BreakAgreement (install per-party inconsistent triples while outputting detection evidence). See Functionality 5.

Algorithm 7 $\mathcal{P}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$ – realization of $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$ in the $(\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle, \mathcal{F}^{\text{acc}}_{\text{co-sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle)$-hybrid model

Use:
Random Extraction parameter: $E = (n - t - t_{acc}) \in \mathbb{N}$
Packing and Batching parameters: $P_1, K_1 \in \mathbb{N}$, $K_2 = P_1K_1$, $P_2 = 1$.
$\forall x \in \{1, 2\} : \mathcal{F}^{\text{acc},x}_{\text{pbhacss}}$ parametrized with P_x, K_x ;
$\mathcal{F}^{\text{acc}}_{\text{sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle$ and $\mathcal{F}^{\text{acc}}_{\text{sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$.

Generation phase

upon request $(\mathsf{Gen.enable}, \mathsf{sid})$:
  Invoke $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle.(\mathsf{Gen.enable}, \mathsf{sid})$
  $\forall d \in \{a, b, c\}$: Invoke $\mathcal{F}^{\text{acc}}_{\text{sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle.(\mathsf{Gen.enable}, \mathsf{sid}^{r_d})$

upon $\mathcal{F}^{\text{acc}}_{\text{sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle.(\mathsf{Gen.ack}, \mathsf{sid})$ ▷ triples are stored under $(\mathsf{sid}^a, \mathsf{sid}^b, \mathsf{sid}^c)$
  $\wedge\ \mathcal{F}^{\text{acc}}_{\text{sr}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle.(\mathsf{Gen.ack}, \mathsf{sid}^{r_d})$ for each $d \in \{a, b, c\}$, **do**:
  $\forall (d, k, j) \in \{a, b, c\} \times [K_1] \times [E]$:
    $\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}.(\mathsf{add}, \mathsf{sid}^{m_d}_{(k,j)}, [\mathsf{sid}^d_{(k,j)}, \mathsf{sid}^{r_d}_{(k,j)}], [1, -1])$. ▷ Shared mask
  $\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}.(\mathsf{reconstruct}, [\ldots, \mathsf{sid}^{m_d}_{(k,j)}, \ldots])$ with $(d, k, j) \in \{a, b, c\} \times [K_1] \times [E]$. ▷ Batched opening

upon $\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}.(\mathsf{retrieve}, \mathsf{sid}^{m_d}_{(k,j)}, \boldsymbol{m}^d_{(k,j)})$, $\forall (d, k, j) \in \{a, b, c\} \times [K_1] \times [E]$, **do**:
  $\forall (d, k, q, j) \in \{a, b, c\} \times [K_1] \times [P_1] \times [E]$:
    $k' = k + q(K_1 - 1)$
    $\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}.(\mathsf{add}, \mathsf{sid}^d_{(k',j)}, [\mathsf{sid}_{\text{unity}}, \mathsf{sid}^{r_d}_{(k',j)}], [\boldsymbol{m}^d_{(k,j)}[q], 1])$ ▷ Add the mask
  trigger notification $(\mathsf{Gen_ack}, \mathsf{sid})$:

Accountability

Upon notification $(\mathsf{detection}, \mathsf{sid}_*, G, \pi)$ from any underlying $\mathcal{F}^{\text{acc}}_{\text{pbhacss}}$ instance:
  trigger notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$.

Again, for clarity of exposition, we omit the published metadata and the associated linear adversarial biasing mechanism (cf. *metadata biasing mechanism* in the previous section)).

The migration (Algorithm 7) is straightforward: (i) Beaver triples are generated via $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle$, (ii) co-shared randomness is generated via $\mathcal{F}^{\text{acc}}_{\text{co-sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$ (to create masks r_a, r_b, r_c in both schemes), (iii) mask opening in scheme 1 followed by (iv) re-sharing into scheme 2 by subtracting/adding the co-shared masks. Special cares are required to handle the different packing and batching parameters.

Theorem 9 (Mask-Opening)**.** *$\mathcal{P}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$ (Algorithm 7) realizes $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$ (Functionality 5) in the $\big(\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle, \mathcal{F}^{\text{acc}}_{\text{co-sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle\big)$-hybrid model.*

Let $\mathcal{A}$ be an arbitrary real-world adversary for $\mathcal{P}^{\text{acc}}_{\text{btg}}\langle\ \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$. We construct a simulator $\mathcal{S}$ interacting with the ideal functionality $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\ \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$ and running a copy $\mathcal{A}^{\text{sim}}|\mathcal{P}^{\text{acc,sim}}_{\text{btg}}$ of the real-world internally. $\mathcal{S}$ also has access to the same hybrid subroutines as the real protocol, namely $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle$ and $\mathcal{F}^{\text{acc}}_{\text{co-sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$. It simulates all honest parties' code of Algorithm 7 for $\mathcal{A}$.

External interface simulation.

- When $\mathcal{S}$ receives $(\mathsf{Gen.enable}, \mathsf{sid}, p_i)$ from $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$, it forwards request $(\mathsf{Gen.enable}, \mathsf{sid})$ to the corresponding simulated real-world process $\bar{p}^{\text{sim}}_i$ in $\mathcal{P}^{\text{acc,sim}}_{\text{btg}}$. Conversely, when $\bar{p}^{\text{sim}}_i$ triggers $(\mathsf{Gen.ack}, \mathsf{sid})$, $\mathcal{S}$ orders $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$ to trigger the delayed output $(\mathsf{Gen.ack}, \mathsf{sid})$ to the dummy process p_i.
- When $\mathcal{A}^{\text{sim}}$ sends $(\mathsf{BreakCompleteness}, \mathsf{sid})$, $\mathcal{S}$ forwards $(\mathsf{BreakCompleteness}, \mathsf{sid})$ to $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$.
- The simulator $\mathcal{S}$ maintains three mappings $\mu_1, \mu_{co}, \tilde{\mu} : [n] \to [n]$, all initially defined as the constant map $i \mapsto 1$. The mappings μ_1 and μ_{co} correspond to the partitionings induced by $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle$ and $\mathcal{F}^{\text{acc}}_{\text{co-sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$, respectively. Whenever either μ_1 or μ_{co} is updated, $\mathcal{S}$ recomputes $\tilde{\mu}$ so that

$$\tilde{\mu}(i) = \tilde{\mu}(j) \iff (\mu_1(i), \mu_{co}(i)) = (\mu_1(j), \mu_{co}(j)).$$

  That is, the partition induced by $\tilde{\mu}$ is always the common refinement of the partitions induced by μ_1 and μ_{co}.
  If $\mathcal{A}^{\text{sim}}$ sends $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mu_1)$ to $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}\rangle$ or $(\mathsf{BreakAgreement}\ , \mathsf{sid}, \boldsymbol{\pi}, \mu_{co})$ to $\mathcal{F}^{\text{acc}}_{\text{co-sr}}\langle\mathcal{F}^{\text{acc},1}_{\text{pbhacss}}, \mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$, then $\mathcal{S}$ updates $(\mu_1, \mu_{co}, \tilde{\mu})$ as described above and forwards $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \tilde{\mu})$ to $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$.
  Later, when a simulated process $\bar{p}^{\text{sim}}_i$ triggers $(\mathsf{detection}, \mathsf{sid}, G_i, \pi_i)$, $\mathcal{S}$ triggers the corresponding delayed output $(\mathsf{detection}, \mathsf{sid}, G_i, \pi_i)$ to the dummy process p_i via $\mathcal{F}^{\text{acc}}_{\text{btg}}\langle\mathcal{F}^{\text{acc},2}_{\text{pbhacss}}\rangle$.

  Case analysis.. We distinguish the nominal and degraded regimes.

Case 1: Nominal regime ($f \leq t$).. In this regime, $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ never executes commands of the form BreakCompleteness or BreakAgreement, and thus simply installs globally consistent triples.

In the simulated real execution, $\mathcal{S}$ runs the protocol $\mathcal{P}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ honestly for all honest parties:

1) It invokes $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}\rangle$ to generate triples (a, b, c) in scheme 1.
2) It invokes $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1},\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ to generate co-shared masks r_a, r_b, r_c in both schemes.
3) It opens masked values $m_d := d - r_d$ in scheme 1 (batched), obtains cleartexts m_d, and installs the scheme 2 sharings as $d = m_d + r_d$ in $\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}$.

We note $(a, b, c = a \otimes b)$ the triples uniformly sampled via $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}\rangle$ in scheme 1. For each $d \in \{a, b, c\}$, the mask r_d is uniform and independent of d, so opening $m_d = d - r_d$ leaks nothing about d beyond what is already implied by (m_d, r_d), and the resulting scheme 2 shares reconstruct to $d = m_d + r_d$ exactly. Therefore, the triple installed into scheme 2 is distributed exactly as in $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$: uniform a, b and $c = ab$.

Case 2: Degraded agreement with accountability ($f \in (t, t_{acc}]$ and BreakAgreement).. In the real world, disagreement may arise independently in each of the two shared-randomness subroutines, namely in $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}\rangle$ and in $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$. Accordingly, the adversary may trigger two a priori unrelated partitionings $\mu_1, \mu_{co} : [n] \to [n]$ via distinct $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mu_1)$ and $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mu_{co})$ requests to these subroutines. These partitionings capture the disagreement structure induced at the level of shared randomness and co-shared randomness, respectively.

The simulator $\mathcal{S}$ maintains the current values of μ_1 and μ_{co}, both initially defined as the constant map $i \mapsto 1$, and recomputes after each update the mapping $\tilde{\mu} : [n] \to [n]$ defined by

$$\tilde{\mu}(i) = \tilde{\mu}(j) \iff (\mu_1(i), \mu_{co}(i)) = (\mu_1(j), \mu_{co}(j)).$$

That is, the partition induced by $\tilde{\mu}$ is the common refinement of the partitions induced by μ_1 and μ_{co}. Whenever the adversary issues a $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mu_1)$ or $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \mu_{co})$ request to the corresponding shared-randomness subroutine, $\mathcal{S}$ updates $(\mu_1, \mu_{co}, \tilde{\mu})$ accordingly and forwards $(\mathsf{BreakAgreement}, \mathsf{sid}, \boldsymbol{\pi}, \tilde{\mu})$ to $\mathcal{F}_{\text{btg}}^{\text{acc}}$.

By construction, $\tilde{\mu}$ preserves exactly the joint disagreement structure of the real execution: any pair of parties that disagree in either shared-randomness instance are separated under $\tilde{\mu}$, while parties that agree in both remain grouped. Thus, the simulator does not introduce spurious disagreement, nor does it mask any real disagreement.

Finally, whenever $\bar{p}_i^{\text{sim}}$ triggers $(\mathsf{detection}, \mathsf{sid}_*, G_i, \pi_i)$, the simulator orders $\mathcal{F}_{\text{btg}}^{\text{acc}}$ to forward the corresponding $(\mathsf{detection}, \mathsf{sid}, G_i, \pi_i)$ notification to p_i. Therefore, every real-world violation of agreement that is detectable is matched by an ideal detection event with the same culprit set, and the environment's view of disagreement and accountability is indistinguishable from that of the real execution.

Fairness.. Liveness is guaranteed by the liveness of the underlying $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\ \mathcal{F}_{\text{pbhacss}}^{\text{acc},1}\rangle$ and $\mathcal{F}_{\text{co-sr}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},1}, \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ functionalities, and therefore holds as long as no valid request $(\mathsf{BreakCompleteness}, \mathsf{sid})$ is issued by $\mathcal{A}$. If such a request is issued, $\mathcal{S}$ forwards $(\mathsf{BreakCompleteness}, \mathsf{sid})$ to $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\ \mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$, thereby justifying the loss of liveness in the ideal execution.

Conclusion.. We have thus constructed a (perfect) forward simulation from the real world to the ideal world, which makes them perfectly indistinguishable. The simulation preserves fair executions. Therefore, $\mathcal{P}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ realizes $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{pbhacss}}^{\text{acc},2}\rangle$ as claimed.

Appendix F.
Beaver triples from a transparent setup only

In this section, we show how to realize the Beaver triple generation functionality $\mathcal{F}_{\text{btg}}^{\text{acc}}\langle\mathcal{F}_{\text{hacss}}^{\text{acc}}(\mathsf{Mt}^P)\rangle$, and thus $\mathcal{F}_{\text{ampc}}^{\text{acc}}$, with a transparent setup (bulletin-board PKI and a CRS) only, i.e, without assuming a threshold-encryption functionality. Of course, the threshold-encryption functionality can be then realized on top of this $\mathcal{F}_{\text{ampc}}^{\text{acc}}$'s realization, at a cost independent of the circuit to be evaluated.

HACSS Compared to the linear realization of $\mathcal{F}_{\text{pbhacss}}^{\text{acc}}(\mathsf{meta}^{\mathsf{AH}})$, $\mathcal{F}_{\text{hacss}}^{\text{acc}}(\mathsf{Mt}^P)$ requires the dealer to *directly* reliably broadcast the matrices $\hat{\boldsymbol{\Phi}}$ of encrypted shares and $\hat{\boldsymbol{\Theta}}$ of opening proofs.

The metadata associated to $\mathcal{F}_{\text{hacss}}^{\text{acc}}(\mathsf{Mt}^P)$ is defined by the map

$$\mathsf{Mt}^P : \big(s,\ (\tilde{\boldsymbol{\xi}}, \mathbf{R})\big) \ \mapsto\ (\boldsymbol{c}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}),$$

where $\boldsymbol{\xi} = (s) :: \tilde{\boldsymbol{\xi}}$ with $\tilde{\boldsymbol{\xi}} = (\xi_2, \ldots, \xi_{\ell+1})$, $\boldsymbol{\phi} = \mathsf{eval}_{\boldsymbol{\alpha}}(\boldsymbol{\xi})$, $\mathbf{R} = (\mathbf{r_e}, \mathbf{r_c}, \mathbf{r}_\theta)$ and $\forall j \in [n]$,

$$\begin{aligned}
\boldsymbol{c}[j] &= \mathsf{Com}(\boldsymbol{\phi}[j]; \boldsymbol{r_c}[j]),\\
\hat{\boldsymbol{\phi}}[j] &= \mathsf{Enc}_{\mathsf{pk}_j}\big(\boldsymbol{\phi}[j];\ \mathbf{r}_e[j]\big),\\
\hat{\boldsymbol{\theta}}[j] &= \mathsf{Enc}_{\mathsf{pk}_j}\big(\boldsymbol{r_c}[j];\ \mathbf{r}_\theta[j]\big).
\end{aligned}$$

Its (costly) realization is straightforward: upon request $\mathcal{F}_{\text{hacss}}^{\text{acc}}(\mathsf{Mt}^P).(\mathsf{share}, \mathsf{sid}_D, s)$, the dealer D invokes

$$\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}_{\mathsf{Mt}^P}).\Big(\mathsf{Prove}, \mathsf{sid}_D, (s, (\tilde{\boldsymbol{\xi}}, \mathbf{R})), \mathsf{Mt}^P\big(s, (\tilde{\boldsymbol{\xi}}, \mathbf{R})\big)\Big),$$

with random witness $(\tilde{\boldsymbol{\xi}}, \mathbf{R})$. Upon notification $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}_{\mathsf{Mt}^P}).(\mathsf{Verify}, \mathsf{sid}_D, D, (\boldsymbol{c}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}))$, process p_j decrypts its provable opening $(\phi(\alpha_j), \theta_j) = \big(\mathsf{Dec}_{\mathsf{sk}_j}(\hat{\boldsymbol{\phi}}[j]), \mathsf{Dec}_{\mathsf{sk}_j}(\hat{\boldsymbol{\theta}}[j])\big)$ and the opening proof triggers reception notification.

This allows a simple shared randomness protocol, described in Algorithm 8.

Algorithm 8 Realizations of $\mathcal{F}_{\text{bhacss}}^{\text{acc}}(\mathsf{Mt}^P)$ and $\mathcal{F}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{bhacss}}^{\text{acc}}(\mathsf{Mt}^P)\rangle$

Parameters:
$n, t, K \in \mathbb{N}$; reconstruction threshold $\ell \in [t,\ n-t)$.
Public Key Encryption $(\mathsf{Enc}, \mathsf{Dec})$ with established setup: public keys $\{\mathsf{pk}_j\}_{j\in[n]}$, secret keys $\{\mathsf{sk}_j\}_{j\in[n]}$. Plaintext space PT and cipher space CT.
Commitment scheme $(\mathsf{Com}, \mathsf{Open}, \mathsf{Verify})$ over $\mathbb{F}_p$.
Each function in $\{\mathsf{Enc}, \mathsf{PC.Com}, \mathsf{PC.Open}\}$ is a $\mathbb{F}_p$-linear homomorphism between abelian groups.

Protocol: $\mathcal{P}_{\text{bhacss}}^{\text{acc}}(\mathsf{Mt}^P)$ **Realize:** $\mathcal{F}_{\text{bhacss}}^{\text{acc}}(\mathsf{Mt}^P)$ **Use:** $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}_{\mathsf{Mt}^P})$

(Batched) Sharing
upon request $(\mathsf{Sh.share}, \mathbf{sid}_D, \boldsymbol{s} = [s_1, \ldots, s_K])$ at $p_i = D$:
  $\mathbf{W_s} = (\tilde{\boldsymbol{\Xi}}, \mathbf{R}_e, \mathbf{R}_c, \mathbf{R}_\theta) \leftarrow_{\$} \tilde{R}^{(K\times\ell)} \times \mathsf{Rand}_{\mathsf{Com}}^{(K\times n)} \times (\mathsf{Rand}_{\mathsf{Enc}}^{(K\times n)})^2$.
  compute $(\mathbf{C}, \hat{\boldsymbol{\Theta}}, \hat{\boldsymbol{\Phi}}) \leftarrow \mathsf{Privil}(\mathbf{s}, \mathbf{W_s})$, and **trigger** $(\mathsf{Sh.witness}, \mathbf{sid}_D, \mathbf{W_s})$.
  $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}_{\mathsf{Mt}^P}).\big(\mathsf{prove}, \mathsf{sid}_D, (\mathbf{s}, \mathbf{W_s}), (\mathbf{C}, \hat{\boldsymbol{\Theta}}, \hat{\boldsymbol{\Phi}})\big)$.

upon $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}_{\mathsf{Mt}^P}).(\mathsf{verify}, \mathsf{sid}_D, D, (\mathbf{C}, \hat{\boldsymbol{\Theta}}, \hat{\boldsymbol{\Phi}}))$:
  $\forall k \in [K]$, $share[\mathsf{sid}_{(D,k)}] \leftarrow (\mathbf{C}[k,:], \hat{\boldsymbol{\Theta}}[k,:], \hat{\boldsymbol{\Phi}}[k,:])$.
  trigger $(\mathsf{Sh.receive}, \mathbf{sid}_D, (\mathbf{C}, \hat{\boldsymbol{\Theta}}, \hat{\boldsymbol{\Phi}}))$.

Reconstruction (dual-threshold)
upon request $(\mathsf{Rec.enable}, \mathsf{sid})$ at p_j:
  $(\mathbf{c}^{\mathsf{sid}}, \hat{\boldsymbol{\theta}}, \hat{\boldsymbol{\phi}}) \leftarrow share[\mathsf{sid}]$.
  $(\boldsymbol{\theta}[j], \boldsymbol{\phi}[j]) \leftarrow (\mathsf{Dec}_{\mathsf{sk}_j}(\hat{\boldsymbol{\theta}}[j]), \mathsf{Dec}_{\mathsf{sk}_j}(\hat{\boldsymbol{\phi}})[j])$.
  broadcast $(\mathsf{Rec.enable}, \mathsf{sid}, j, \boldsymbol{\theta}[j], \boldsymbol{\phi}[j])$.

upon receiving $(\mathsf{Rec.enable}, \mathsf{sid}, j, v^j, \theta^j)$, s.t. $\mathsf{Verify}_j(\mathbf{c}^{\mathsf{sid}}, v^j, \theta^j)$:
  $\boldsymbol{\phi}^{\mathsf{sid}}[j] \leftarrow v^j$.
upon $\exists J \subset [n], |J| = \ell, \forall j \in J$, $\boldsymbol{\phi}^{\mathsf{sid}}[j] \neq \perp$:
  $\boldsymbol{\xi}^{\mathsf{sid}}[1] \leftarrow \mathsf{interpol}(\boldsymbol{\phi}_J^{\mathsf{sid}})$, and **trigger notification** $(\mathsf{Rec.retrieve}, \mathsf{sid}, \boldsymbol{\xi}^{\mathsf{sid}}[1])$.

Local addition
upon $(\mathsf{local_add}, \mathsf{sid}_{sum}, \mathbf{sid}, \mathbf{x})$:
  $share[\mathsf{sid}_{sum}] \leftarrow \bigoplus_{k\in[|\mathbf{x}|]}(\mathbf{x}[k] \odot share[\mathbf{sid}[k]])$.

Protocol: $\mathcal{P}_{\text{sr}}^{\text{acc}}(\mathsf{Mt}^P)$ **Realize:** $\mathcal{F}_{\text{sr}}^{\text{acc}}\langle\mathcal{F}_{\text{bhacss}}^{\text{acc}}(\mathsf{Mt}^P)\rangle$
Use: $\mathcal{F}_{\text{bhacss}}^{\text{acc}}(\mathsf{Mt}^P)$, and $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$.

upon request $(\mathsf{Gen.enable}, \mathsf{sid}, B)$ by p_i:
  $\mathbf{s}^i \leftarrow_{\$} \tilde{R}^B$.
  $\mathcal{F}_{\text{bhacss}}^{\text{acc}}(\mathsf{Mt}^P).(\mathsf{share}, \mathsf{sid}_i, \mathbf{s}^i)$.

upon $(\mathsf{Sh.receive}, \mathbf{sid}_j, (\mathbf{C}^j, \hat{\boldsymbol{\Theta}}^j, \hat{\boldsymbol{\Phi}}^j))$:
  $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}.(\mathsf{acs_validate}, \mathsf{sid}, p_j)$.

upon $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}.(\mathsf{acs_decide}, \mathsf{sid}, Q = \{j_1, ..., j_{|Q|}\})$:
  trigger $(\mathsf{local_add}, \mathbf{sid}, [\mathsf{sid}_{j_1}, ..., \mathsf{sid}_{j_{|Q|}}], [1, ..., 1])$
  trigger $\big(\mathsf{Gen.receive}, \mathsf{sid}, \bigoplus_{j\in Q}(\mathbf{C}^j, \hat{\boldsymbol{\Theta}}^j, \hat{\boldsymbol{\Phi}}^j)\big)$.

Accountability
upon notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$ from underlying $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}$, or $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$:
  trigger notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$.

We can now describe the Beaver $\mathcal{F}_{\text{btg}}^{\text{acc}}$'s realization $\mathcal{P}_{\text{btg_privil}}^{\text{acc}}$ (see Algorithm 9), which operates in the $(\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}_{\mathsf{Mt}^P}), \mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}(\mathcal{R}_{\mathsf{prod}^{\mathsf{P}}}), \mathcal{F}_{\text{idx_acs}}^{\text{acc}})$-hybrid model.

Each party p_i can share a random contribution $a^i \in \mathbb{F}$. Once the parties agree (via $\mathcal{F}_{\text{idx_acs}}$) on a core set Q of $n-t$ processes who correctly shared their contributions via $\mathcal{F}_{\text{hacss}}^{\text{acc}}(\mathsf{Mt}^P)$, they sum both the clear text shares and the encrypted shares to obtain a common shared random value a with metadata $meta^a = (\boldsymbol{c}^a, \hat{\boldsymbol{\theta}}^a, \hat{\boldsymbol{\phi}}^a) = \mathsf{Mt}^P(a, *)$:

$$\phi^a(\alpha_j) = \bigoplus_{i\in Q} \phi^{a^i}(\alpha_j), \qquad \hat{\boldsymbol{\phi}}^a = \bigoplus_{i\in Q} \hat{\boldsymbol{\phi}}^{a^i}, \qquad \boldsymbol{c}^a = \bigoplus_{i\in Q} \boldsymbol{c}^{a^i}, \qquad \hat{\boldsymbol{\theta}}^a = \bigoplus_{i\in Q} \hat{\boldsymbol{\theta}}^{a^i}.$$

Next, each process p_i locally samples:

Algorithm 9 $\mathcal{P}^{\mathrm{acc}}_{\mathrm{btg}}(\mathsf{Mt}^P)$ – $\mathcal{F}^{\mathrm{acc}}_{\mathrm{btg}}\langle\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)\rangle$'s realization

Parameters:
$n, t, K \in \mathbb{N}$; reconstruction threshold $\ell \in [t,\ n-t)$.
Public Key Encryption $(\mathsf{Enc}, \mathsf{Dec})$ with established setup: public keys $\{\mathsf{pk}_j\}_{j\in[n]}$, secret keys $\{\mathsf{sk}_j\}_{j\in[n]}$. Plaintext space PT and cipher space CT.
Commitment scheme $(\mathsf{Com}, \mathsf{Open}, \mathsf{Verify})$ over $\mathbb{F}_p$.
Each function in $\{\mathsf{Enc}, \mathsf{PC.Com}, \mathsf{PC.Open}\}$ is a $\mathbb{F}_p$-linear homomorphism between abelian groups.

Use: $\mathcal{F}^{\mathrm{acc}}_{\mathrm{sr}}\langle\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)\rangle$, $\mathcal{F}^{1:\mathrm{M},\mathrm{acc}}_{\mathrm{zkPoK}}(\mathcal{R}_{\mathsf{priviprod}}(*))$, and $\mathcal{F}^{\mathrm{acc}}_{\mathrm{idx_acs}}$.

upon request $(\mathsf{Gen.enable}, \mathsf{sid})$ by p_i:
  $\mathcal{F}^{\mathrm{acc}}_{\mathrm{sr}}\langle\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)\rangle.(\mathsf{Gen.enable}, \mathsf{sid}, 1)$:

upon $\mathcal{F}^{\mathrm{acc}}_{\mathrm{sr}}\langle\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)\rangle.(\mathsf{Gen.receive}, \mathsf{sid}, m^a)$: ▷ $m^a = (\hat{\boldsymbol{\phi}}^a, \mathbf{c}^a, \hat{\boldsymbol{\theta}}^a)$
  sample polynomials $(\xi^{b^i}, \xi^{m^i}) \leftarrow_{\$} \tilde{R}^{(\ell+1)}$, with $\xi^{m^i}[1] = 0$,
  sample $((\mathbf{r}_e^{b^i}, r_c^{b^i}, r_\theta^{b^i}), (\mathbf{r}_e^{m^i}, r_c^{m^i}, r_\theta^{m^i})) \leftarrow_{\$} (\mathsf{Rand}^n_{\mathsf{Enc}} \times \mathsf{Rand}^n_{\mathsf{Com}} \times \mathsf{Rand}^n_{\mathsf{Enc}})^2$.
  Define $\mathbf{r}_w^{(b^i,c^i)} = (\xi^{b^i}, \mathbf{r}_e^{b^i}, r_c^{b^i}, \xi^{m^i}, \mathbf{r}_e^{m^i}, r_c^{m^i})$.
  compute $stmt^{(b^i,c^i)} \leftarrow \mathsf{priviprod}(m^a)(\mathbf{r}_w^{(b^i,c^i)})$.
  invoke $\mathcal{F}^{1:\mathrm{M},\mathrm{acc}}_{\mathrm{zkPoK}}(\mathcal{R}_{\mathsf{priviprod}}(m^a))$ with
    $(\mathsf{prove}, \mathsf{sid}_i, \mathbf{r}_w^{(b^i,c^i)}, stmt^{(b^i,c^i)})$.

upon $\mathcal{F}^{1:\mathrm{M},\mathrm{acc}}_{\mathrm{zkPoK}}(\mathcal{R}_{\mathsf{priviprod}}(m^a))$ delivers
  $(\mathsf{prove}, \mathsf{sid}_j, p_j, (c^{b^j}, \hat{\boldsymbol{\theta}}^{b^j}, \hat{\boldsymbol{\phi}}^{b^j}, c^{(ab^j)}, \hat{\boldsymbol{\theta}}^{(ab^j)}, \hat{\boldsymbol{\phi}}^{(ab^j)}))$:
    $\mathcal{F}^{\mathrm{acc}}_{\mathrm{idx_acs}}.(\mathsf{acs_validate}, \mathsf{sid}_{bc}, p_j)$.

upon $\mathcal{F}^{\mathrm{acc}}_{\mathrm{idx_acs}}.(\mathsf{acs_decide}, \mathsf{sid}_{bc}, Q_{bc})$:
  $(\hat{\boldsymbol{\phi}}^b, c^b, \hat{\boldsymbol{\theta}}^b) \leftarrow \bigoplus_{j\in Q_{bc}} (\hat{\boldsymbol{\phi}}^{b^j}, c^{b^j}, \hat{\boldsymbol{\theta}}^{b^j})$.
  $(\hat{\boldsymbol{\phi}}^c, c^c, \hat{\boldsymbol{\theta}}^c) \leftarrow \bigoplus_{j\in Q_{bc}} (\hat{\boldsymbol{\phi}}^{ab^j}, c^{ab^j}, \hat{\boldsymbol{\theta}}^{ab^j})$.
  trigger $(\mathsf{Gen.receive}, \mathsf{sid})$.
  For each $d \in \{a, b, c\}$:
    $share[\mathsf{sid}_d] \leftarrow (c^d, \mathsf{Dec}_{sk_i}(\hat{\boldsymbol{\phi}}^d[i]), \mathsf{Dec}_{sk_i}(\hat{\boldsymbol{\theta}}^d[i]))$.

upon request $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P).(\mathsf{request}, \mathsf{sid}, *)$ by p_i:
  follow $\mathcal{P}^{\mathrm{acc}}_{\mathrm{sr_privil}}$ (Algorithm 8) accordingly with joint $share[]$ variable.

Accountability
upon notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$
  from $\mathcal{F}^{\mathrm{acc}}_{\mathrm{sr}}$, $\mathcal{F}^{1:\mathrm{M},\mathrm{acc}}_{\mathrm{zkPoK}}$, or $\mathcal{F}^{\mathrm{acc}}_{\mathrm{idx_acs}}$:
  trigger notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$.

- a fresh polynomial $\boldsymbol{\xi}^{b^i}$, contributing to a second random value, with $b_i = \boldsymbol{\xi}^{b^i}[1]$,
- fresh randomness $(\mathbf{R}^{b^i} = (\mathbf{r}_c^{b^i}, \mathbf{r}_e^{b^i}, \mathbf{r}_\theta^{b^i})$ for its commitment, its encryption, and the encryption of its commitment randomness.
- coefficients $\tilde{\boldsymbol{\xi}}^{m^i} = (\xi_2^{m^i}, \ldots, \xi_{\ell+1}^{m^i})$ of a masking polynomial $\boldsymbol{\xi}^{m^i} = (0) :: \tilde{\boldsymbol{\xi}}^{m^i}$ with $\boldsymbol{\xi}^{m^i}[1] = 0$, together with similar randomness $\mathbf{R}^{m^i} = (\mathbf{r}_c^{m^i}, \mathbf{r}_e^{m^i}, \mathbf{r}_\theta^{m^i})$ encryption randomness.

It then compute

$$\big((\boldsymbol{c}^{b^i}, \hat{\boldsymbol{\theta}}^{b^i}, \hat{\boldsymbol{\phi}}^{b^i}), (\boldsymbol{c}^{(ab^i)}, \hat{\boldsymbol{\theta}}^{(ab^i)}, \hat{\boldsymbol{\phi}}^{(ab^i)})\big) \ \leftarrow\ \mathsf{prod}^{\mathsf{P}}_{meta^a}\big(\boldsymbol{\xi}^{b^i},\ (\mathbf{R}^{b^i}, \mathbf{R}^{m^i}),\ \tilde{\boldsymbol{\xi}}^{m^i}\big)\big),$$

where $\mathsf{prod}^{\mathsf{P}}$ is a hard-to-invert but easy-to-compute homomorphism between abelian groups.

Concretely, $(\boldsymbol{c}^{b^i}, \hat{\boldsymbol{\theta}}^{b^i}, \hat{\boldsymbol{\phi}}^{b^i}) = \mathsf{Mt}^P\big(\boldsymbol{\xi}^{b^i}, \mathbf{R}^{b^i}\big)$, $(\boldsymbol{c}^{m^i}, \hat{\boldsymbol{\theta}}^{m^i}, \hat{\boldsymbol{\phi}}^{m^i}) = \mathsf{Mt}^P\big(\boldsymbol{\xi}^{m^i}, \mathbf{R}^{m^i}\big)$, while the product contribution is defined as:

$(\boldsymbol{c}^{ab^i}, \hat{\boldsymbol{\theta}}^{ab^i}, \hat{\boldsymbol{\phi}}^{ab^i}) = \big(b^i \odot_{\mathsf{Mt}^P} (\boldsymbol{c}^a, \hat{\boldsymbol{\theta}}^a, \hat{\boldsymbol{\phi}}^a)\big) \oplus_{\mathsf{Mt}^P} (\boldsymbol{c}^{m^i}, \hat{\boldsymbol{\theta}}^{m^i}, \hat{\boldsymbol{\phi}}^{m^i})$.

$(\boldsymbol{c}^{ab^i}, \hat{\boldsymbol{\theta}}^{ab^i}, \hat{\boldsymbol{\phi}}^{ab^i}) = \mathsf{Mt}^P\Big((b^i \odot a), \big(b^i \odot (\tilde{\xi}^a, \mathbf{R}^a) \oplus (\tilde{\xi}^{m^i}, \mathbf{R}^{m^i})\big)\Big)$

Finally, p_i disseminates $\big((\boldsymbol{c}^{b^i}, \hat{\boldsymbol{\theta}}^{b^i}, \hat{\boldsymbol{\phi}}^{b^i}), (\boldsymbol{c}^{(ab^i)}, \hat{\boldsymbol{\theta}}^{(ab^i)}, \hat{\boldsymbol{\phi}}^{(ab^i)})\big)$ via $\mathcal{F}^{1:\mathrm{M},\mathrm{acc}}_{\mathrm{zkPoK}}(\mathcal{R}_{\mathsf{prod}^{\mathsf{P}}_{meta^a}})$, thereby sharing contributions for both the second and third components.

The parties then agree (via $\mathcal{F}_{\mathrm{idx_acs}}$) on a second core set Q' of $n-t$ processes who correctly shared their second and third contributions. By summing over Q', they obtain shares of

$$\phi^b(\alpha_j) = \bigoplus_{i\in Q'} \phi^{b^i}(\alpha_j), \qquad \phi^c(\alpha_j) = \bigoplus_{i\in Q'} \phi^{(ab^i)}(\alpha_j),$$

with corresponding commitments and opening proofs:

$$\boldsymbol{c}^b = \bigoplus_{i\in Q'} \boldsymbol{c}^{b^i}, \qquad \boldsymbol{c}^c = \bigoplus_{i\in Q'} \boldsymbol{c}^{ab^i}, \qquad \hat{\boldsymbol{\theta}}^b = \bigoplus_{i\in Q'} \hat{\boldsymbol{\theta}}^{b^i}, \qquad \hat{\boldsymbol{\theta}}^c = \bigoplus_{i\in Q'} \hat{\boldsymbol{\theta}}^{ab^i},$$

This yields shares of a Beaver triple (a, b, c) over $\mathbb{F}$ with $c = a \otimes_{\mathbb{F}} b$, and metadata $\mathsf{Mt}^P(d, *)$ for each $d \in \{a, b, c\}$.

Theorem 10 (Privileged-BTG)**.** *(Theorem 3) Protocol $\mathcal{P}^{\mathrm{acc}}_{\mathrm{btg}}(\mathsf{Mt}^P)$ Algorithm 9 realizes $\mathcal{F}^{\mathrm{acc}}_{\mathrm{btg}}\langle\mathcal{F}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}(\mathsf{Mt}^P)\rangle$ of Functionality 5 in the $\big(\mathcal{F}^{\mathrm{acc}}_{\text{idx-acs}}, \mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}\big)$-hybrid model with $O(\kappa n^3)$ communication per triple, and $O(1)$ rounds.*

Proof. Let $\mathcal{A}$ be a real-world adversary interacting with $\mathcal{P}^{\mathrm{acc}}_{\mathrm{btg}}(\mathsf{Mt}^P)$, in short $\mathcal{P}^{\mathrm{acc}}_{\mathrm{btg}}$. Let $\mathcal{S}$ be the corresponding simulator interacting with $\mathcal{F}^{\mathrm{acc}}_{\mathrm{btg}}\langle\mathcal{F}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}(\mathsf{Mt}^P)\rangle$, in short $\mathcal{F}^{\mathrm{acc}}_{\mathrm{btg}}$. $\mathcal{S}$ simulates a copy of the real world.

Simulating $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)$. Since $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$ is an ideal subroutine, $\mathcal{S}$ can: (i) write statements on behalf of honest parties to the statement interface; (ii) read the witness–statement pairs submitted by corrupted parties from the witness interface. This makes simulation of $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)$ immediate: whenever an honest dealer D triggers a sharing in $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)$ (line 5– 8), $\mathcal{S}$ receives the corresponding metadata and writes it directly on the statement's tape of $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$. Conversely, whenever a simulated real-world honest process is notified of a sharing from a corrupted dealer with associated metadata $m = \mathsf{Mt}^P(s, r)$, the associated witness (s, r) must have been written on the corresponding $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$'s tape. The simulator can read it and feed $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)$ accordingly.

Simulating $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)$. Whenever an honest dealer D triggers a sharing (line 5– 8) in $\mathcal{F}^{\mathrm{acc}}_{\mathrm{bhacss}}(\mathsf{Mt}^P)$, $\mathcal{S}$ receives the corresponding metadata and writes it directly on the statement's tape of $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$.

Simulating shared randomness generation. In the real protocol, each party contributes a random secret and the parties aggregate the corresponding metadata over a core set returned by $\mathcal{F}^{\mathrm{acc}}_{\text{idx-acs}}$ (line 23– 30). $\mathcal{S}$ simulates the entire protocol execution and samples all honest contributions uniformly. Corrupted contributions are extracted from the witness tape of $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$. The adversary (via $\mathcal{F}^{\mathrm{acc}}_{\text{idx-acs}}$) determines the decided core set Q, which in turn determines which contributions are aggregated. This captures the fact that the adversary may correlate its own contributions (and thus bias the *distribution* of the resulting metadata) with the already observed honest statements, while the resulting secret value remains uniform as long as the decided core set contains at least one honest contribution. For simplicity, the ideal shared-randomness functionality abstracts away this metadata-biasing machinery and specifies only the distribution of the resulting secret.

Simulating Beaver triple generation. $\mathcal{S}$ first simulates the call to shared randomness and obtains a statement m^a (line 7) corresponding to a uniformly distributed shared a. Then each party locally samples (b^i, m^i) (and additional randomness), and proves, via $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$, that its published pair of metadata values for b^i and $ab^i + m^i$ is consistent with the public m^a (line 11– 13). Honest parties can be entirely simulated. For corrupted parties, $\mathcal{S}$ reads the corresponding witnesses/statements from the witness tape. The adversary again influences which contributions are aggregated through $\mathcal{F}^{\mathrm{acc}}_{\text{idx-acs}}$ (line 15– 17), which may bias the aggregate metadata, but not the induced secrets.

Correctness. By correctness of $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$ for $\mathcal{R}_{\mathsf{Mt}^P}$ and $\mathcal{R}_{\mathsf{priviprod}}(m^a)$, every validated contribution is well-formed. By additive homomorphism of Mt (and componentwise homomorphism of the commitment and encryption maps), the aggregated metadata corresponds to the sum of the underlying secrets. Since the decided sets have size $n - t$ and $f < n - t$, each such set contains at least one honest contribution; hence the aggregated secrets a and b are uniformly distributed in $\mathbb{F}$, and the aggregated third component satisfies $c = a \otimes_{\mathbb{F}} b$ by the product relation and masking with zero-constant polynomials. Therefore the output of $\mathcal{P}^{\mathrm{acc}}_{\mathrm{btg}}$ is a correct Beaver triple sharing as required by $\mathcal{F}^{\mathrm{acc}}_{\mathrm{btg}}\langle\mathcal{F}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}(\mathsf{Mt}^P)\rangle$.

Liveness. Liveness follows from liveness of the underlying ideal subroutines $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$ and $\mathcal{F}^{\mathrm{acc}}_{\text{idx-acs}}$.

Accountability. Accountability follows from the accountable $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$ and $\mathcal{F}^{\mathrm{acc}}_{\text{idx-acs}}$. The simulator maintains a partition of the processes that is consistent with the partitionings induced by the underlying accountable subroutines. Concretely, two processes belong to the same simulator partition as long as they belong to the same partition in each invoked subroutine instance. Whenever a subroutine outputs a detection notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$, the simulator updates its global partition accordingly and forwards the notification to the ideal functionality.

Within each resulting partition, the Beaver triples remain well-defined, uniform, and correct: aggregation and homomorphic computation are performed independently inside each partition, and correctness of the NP relations ensures that $c = a \otimes_{\mathbb{F}} b$ holds partition-wise. Uniformity is preserved partition-wise as well, since in every partition of size greater than t, at least one honest contribution remains, and therefore the aggregated secrets retain their one-time-pad uniformity. This matches the behavior prescribed by $\mathcal{F}^{\mathrm{acc}}_{\mathrm{btg}}\langle\mathcal{F}^{\mathrm{AH,acc}}_{\mathrm{pbhacss}}(\mathsf{Mt}^P)\rangle$ under $\mathsf{BreakAgreement}$. Any notification $(\mathsf{detection}, \mathsf{sid}, G, \pi)$ produced by $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$ or $\mathcal{F}^{\mathrm{acc}}_{\text{idx-acs}}$ is forwarded unchanged by honest parties (line 33, line 27).

Complexity. Each Beaver triple involves $3n$ metadata objects of the form $m = \mathsf{Mt}^{\mathsf{P}}(s, r)$ that are certified via $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$, together with two invocations of $\mathcal{F}^{\mathrm{acc}}_{\text{idx_acs}}$.

Instantiating $\mathcal{F}^{\mathrm{1:M,acc}}_{\mathrm{zkpok}}$ using the UC-secure NIZK compiler of [47], each proof corresponds to a NIZK π of size $O(\kappa n)$, where κ denotes the group / ciphertext security parameter and n is the number of witness components in the linear relation.

Disseminating such a proof via $\mathcal{F}_{\mathrm{rbc}}^{\mathrm{acc}}$ requires $O(\kappa n^2)$ communication. Since $3n$ such certified metadata objects are produced per Beaver triple, the total communication cost per triple is $O(\kappa n^3)$.

This concludes the proof. □

Appendix G. Homomorphic Polynomial Commitment.

Trapdoor Polynomial Commitment Scheme Interface.. We adapt the presentation of [57]. Fix a ring R being a $\mathbb{F}$-vector space and a vector of pairwise distinct evaluation points $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}^n$. Let $\mathsf{R}_\ell[X]$ denote the set of polynomials over R of degree at most $\ell \geq 1$.

A *trapdoor polynomial commitment scheme* over $(\mathsf{R}, \boldsymbol{\alpha})$ is a tuple

$$\mathsf{TPC} = (\mathsf{Gen}, \mathsf{Com}, \mathsf{TCom}, \mathsf{Open}, \mathsf{TOpen}, \mathsf{Ver}, \mathsf{TCol}, \mathsf{TSim})$$

with the following PPT (or DPT) algorithms:

- $\mathsf{Gen}(1^\kappa)$: a PPT algorithm that takes as input the security parameter 1^κ and outputs a pair (srs, τ), where srs is a public structured reference string and τ is a trapdoor associated with srs.
- $\mathsf{Com}^{\mathsf{srs}}(\phi; r)$: a PPT algorithm that takes as input the SRS srs and a polynomial $\phi \in \mathsf{R}_\ell[X]$, samples randomness $r \leftarrow \mathsf{COIN}^{\mathsf{com}}$, and outputs a commitment
$$c = \mathsf{Com}^{\mathsf{srs}}(\phi; r).$$
- $\mathsf{TCom}^{\tau}(1^\kappa)$: a PPT algorithm that takes as input the trapdoor τ and outputs a dummy commitment together with simulation state:
$$(c^\diamond, \sigma^\diamond) \leftarrow \mathsf{TCom}^{\tau}(1^\kappa).$$
- $\mathsf{Open}^{\mathsf{srs}}_{\alpha_i}(\phi; r)$: a PPT algorithm that takes as input srs, a polynomial $\phi \in \mathsf{R}_\ell[X]$, randomness $r \in \mathsf{COIN}^{\mathsf{com}}$ used in the commitment, and an evaluation point $\alpha_i \in \{\alpha_1, \ldots, \alpha_n\}$, and outputs an opening proof for the evaluation $\phi(\alpha_i)$ at point α_i:
$$\theta_i = \mathsf{Open}^{\mathsf{srs}}_{\alpha_i}(\phi; r).$$
We note $\mathsf{EOpen}^{\mathsf{srs}}_{\alpha_i}(\phi; r) = \big(\phi(\alpha_i), \mathsf{Open}^{\mathsf{srs}}_{\alpha_i}(\phi; r)\big)$.
- $\mathsf{TOpen}^{\tau}_{\alpha_i}(c^\diamond, \sigma^\diamond)$: a PPT algorithm that takes as input the trapdoor τ, a dummy commitment/state pair $(c^\diamond, \sigma^\diamond)$ output by TCom^{τ}, an evaluation point α_i, and outputs a simulated opening pair
$$(v_i^\diamond, \theta_i^\diamond) = \mathsf{TOpen}^{\tau}_{\alpha_i}(c^\diamond, \sigma^\diamond).$$
- $\mathsf{Ver}^{\mathsf{srs}}_{\alpha_i}(c, v_i, \theta_i)$: a DPT algorithm that takes as input srs, a commitment c, a point α_i, a value $v_i \in \mathsf{R}$, and an opening proof θ_i, and outputs 1 if and only if θ_i is a valid opening of c at point α_i with value v_i.
- $\mathsf{TCol}^{\tau}\big(c^\diamond, \sigma^\diamond, \{(\alpha_i, v_i^\diamond, \theta_i^\diamond)\}_{i\in J}, s\big)$: a DPT algorithm that takes as input the trapdoor τ, a dummy pair $(c^\diamond, \sigma^\diamond)$, a set of pre-simulated openings $\{(\alpha_i, v_i^\diamond, \theta_i^\diamond)\}_{i\in J}$ for some index set $J \subseteq [n]$ of size $|J| \leq \ell$, and a target constant term $s \in \mathsf{R}$. It outputs a pair
$$(\phi^*, r^*) \leftarrow \mathsf{TCol}^{\tau}\big(c^\diamond, \sigma^\diamond, \{(\alpha_i, v_i^\diamond, \theta_i^\diamond)\}_{i\in J}, s\big)$$
where $\phi^* \in \mathsf{R}_\ell[X]$ and $r^* \in \mathsf{COIN}^{\mathsf{com}}$ satisfy:
  1) $\phi^*(0) = s$,
  2) $c^\diamond = \mathsf{Com}^{\mathsf{srs}}(\phi^*; r^*)$,
  3) $\forall i \in J$: $(v_i^\diamond, \theta_i^\diamond) = \big(\phi^*(\alpha_i), \mathsf{Open}^{\mathsf{srs}}_{\alpha_i}(\phi^*; r^*)\big)$, i.e. the pre-simulated openings are exactly equal to the honest openings of $(\phi^*; r^*)$.
- $\mathsf{TSim}^{\tau}(\phi, r)$: a PPT algorithm that takes as input the trapdoor τ, a polynomial $\phi \in \mathsf{R}_\ell[X]$, randomness $r \in \mathsf{COIN}^{\mathsf{com}}$ used in the commitment and outputs a simulation state $\sigma^\diamond = \mathsf{TSim}^{\tau}(\phi, r)$. Then, applying TCol with $\sigma^\diamond$ and correctly generated material for commitment and openings, allows returning ϕ, exactly as in the equivocal mode.

Moreover, we say that TPC is *randomness-free* if $\mathsf{Com}^{\mathsf{srs}}$ takes only the committed polynomial as an input (the randomness is ignored).

We note $\mathsf{TPC.Share}^{\mathsf{srs}}_{\boldsymbol{\alpha}}(\boldsymbol{\xi})$ the function taking $\boldsymbol{\xi}$ as input parameter and returning $(\boldsymbol{\phi}, c, \boldsymbol{\theta})$, such that $c = \mathsf{TPC.Com}^{\mathsf{srs}}(\boldsymbol{\xi})$, $\boldsymbol{\phi} = \mathsf{eval}_{\boldsymbol{\alpha}}(\boldsymbol{\xi})$, and $\forall i \in [n]$, $\boldsymbol{\theta}[i] = \mathsf{TPC.Open}^{\mathsf{srs}}_{\boldsymbol{\alpha}[i]}(\boldsymbol{\xi})$. Similarly, we note $\mathsf{TPC.Share}^{\mathsf{srs}}_{\boldsymbol{\alpha},\mathbf{pk}}(\boldsymbol{\xi}, \boldsymbol{r}_e) = (\hat{\boldsymbol{\phi}}, c, \hat{\boldsymbol{\theta}})$ such that $(\boldsymbol{\phi}, c, \boldsymbol{\theta}) = \mathsf{TPC.Share}^{\mathsf{srs}}_{\boldsymbol{\alpha}}(\boldsymbol{\xi})$ and $\forall i \in [n]$, $\hat{\boldsymbol{\phi}}[i] = \mathsf{Enc}_{\mathbf{pk}[i]}(\boldsymbol{\phi}[i], \boldsymbol{r}_e[i])$ and $\hat{\boldsymbol{\theta}}[i] = \mathsf{Enc}_{\mathbf{pk}[i]}(\boldsymbol{\theta}[i], \boldsymbol{r}_e[i+n])$

Security.. Let $(\mathsf{srs}, \tau) \leftarrow \mathsf{Gen}(1^\kappa)$ be the public parameters. A trapdoor polynomial commitment scheme is secure if it satisfies:

- *Completeness*: Honestly generated commitments and proofs always verify correctly.
- *Evaluation Binding*: No adversary can produce a valid commitment c and two different valid openings (v_i, θ_i), (v_i', θ_i') for the same evaluation point α_i with $v_i \neq v_i'$.
- *Hiding.* For any ℓ distinct positions $J' \subseteq [n]$ with $|J'| = \ell$, given a commitment c and valid openings $\{(v_i, \theta_i)\}_{i \in J'}$, the value $\phi(\alpha^*)$ at any other point $\alpha^* \in \mathbb{F}$, $\alpha^* \notin \{\alpha_i : i \in J'\}$, remains computationally hidden.

Additionally, the scheme must ensure *Equivocality*:

We require that TPC admits equivocal simulation up to ℓ evaluation points.

Let $s \in \mathsf{R}$ be any secret. Let $J \subseteq [n]$ be any index set with $|J| \leq \max(\ell - 1, 0)$. The following two ensembles are computationally indistinguishable in κ.

Real view (honest commitments and openings):

$$\Big\{ (c, \{(v_i, \theta_i)\}_{i \in [n]}) \;\Big|\; \begin{array}{l} (\mathsf{srs}, \tau) \leftarrow \mathsf{Gen}(1^\kappa), \\ \phi \leftarrow_{\$} \{\phi_s \in \mathsf{R}_\ell[X] \mid \phi_s(0) = s\},\ r \leftarrow_{\$} \mathsf{COIN}^{\mathsf{com}}, \\ c = \mathsf{Com}^{\mathsf{srs}}(\phi; r), \\ \forall i \in [n] :\ (v_i, \theta_i) = \mathsf{EOpen}^{\mathsf{srs}}_{\alpha_i}(\phi; r) \end{array} \Big\}_{\kappa \in \mathbb{N}, s \in \mathsf{R}}.$$

Equivocal view (dummy commitment and trapdoor-retrofitted openings):

$$\Big\{ (c^\diamond, \{(v_i^\diamond, \theta_i^\diamond)\}_{i \in [n]}) \;\Big|\; \begin{array}{l} (\mathsf{srs}, \tau) \leftarrow \mathsf{Gen}(1^\kappa), \\ (c^\diamond, \sigma^\diamond) \leftarrow \mathsf{TCom}^\tau(1^\kappa), \\ \forall i \in J :\ (v_i^\diamond, \theta_i^\diamond) = \mathsf{TOpen}^\tau_{\alpha_i}(c^\diamond, \sigma^\diamond), \\ (\phi^*, r^*) \leftarrow \mathsf{TCol}^\tau\big(c^\diamond, \sigma^\diamond, \{(\alpha_i, v_i^\diamond, \theta_i^\diamond)\}_{i \in J}, s\big), \\ \forall i \in [n] \setminus J :\ (v_i^\diamond, \theta_i^\diamond) = \mathsf{EOpen}^{\mathsf{srs}}_{\alpha_i}(\phi^*; r^*) \end{array} \Big\}_{\kappa \in \mathbb{N}, s \in \mathsf{R}}.$$

Intuitively, the real experiment commits and opens honestly to ϕ, whereas the equivocal experiment generates a dummy commitment and simulated openings using τ, and then uses TCol^τ to "explain" that simulated view as a valid commitment to some polynomial ϕ^* with the same constant term $s = \phi(0)$. No efficient distinguisher can tell the experiments apart.

Finally, we require that if:

- $\sigma^\diamond \leftarrow \mathsf{TSim}^\tau(\phi, r)$
- $c = \mathsf{Com}^{\mathsf{srs}}(\phi, r)$
- $\forall i \in [n]$: $\theta_i = \mathsf{Open}^{\mathsf{srs}}_{\alpha_i}(\phi, r)$,

then, $\forall J \subset [n]$, $|J| \leq \ell - 1$, we have $\phi = \mathsf{TCol}^\tau(\sigma^\diamond, c, \{\phi(\alpha_i), \theta_i\}_{i \in J}, \phi(0))$.

Linearity.. We require that TPC preserves $\mathbb{F}$-linearity across commitments, evaluations, and opening proofs. Formally, let $(\mathsf{srs}, \tau) \leftarrow \mathsf{Gen}(1^\kappa)$. For any two polynomials $\phi^1, \phi^2 \in \mathsf{R}_\ell[X]$, any scalars $a, b \in \mathbb{F}$, any index $i \in [n]$, and any randomness $r^1, r^2 \leftarrow \mathsf{COIN}^{\mathsf{com}}$, let

$$c^j = \mathsf{Com}^{\mathsf{srs}}(\phi^j; r^j) \quad \text{and} \quad (v_i^j, \theta_i^j) = \mathsf{EOpen}^{\mathsf{srs}}_{\alpha_i}(\phi^j; r^j) \text{ for } j \in \{1, 2\}.$$

Define

$$\phi := a\phi^1 + b\phi^2, \qquad r := ar^1 + br^2,$$

and

$$\begin{aligned} c &:= (a \odot_{\mathbb{H}_C} c^1) \ \oplus_{\mathbb{H}_C} \ (b \odot_{\mathbb{H}_C} c^2), \\ v_i &:= (a \odot_{\mathsf{R}} v_i^1) \ \oplus_{\mathsf{R}} \ (b \odot_{\mathsf{R}} v_i^2). \\ \theta_i &:= (a \odot_{\mathbb{H}_\theta} \theta_i^1) \ \oplus_{\mathbb{H}_\theta} \ (b \odot_{\mathbb{H}_\theta} \theta_i^2). \end{aligned}$$

Then the following correctness condition must hold:

$$(v_i, \theta_i) = \mathsf{EOpen}^{\mathsf{srs}}_{\alpha_i}(\phi; r),$$

and, therefore,

$$\mathsf{Ver}^{\mathsf{srs}}_{\alpha_i}(c, v_i, \theta_i) = 1.$$

In particular:

- commitments are $\mathbb{F}$-linear in the committed polynomial,
- evaluation values are $\mathbb{F}$-linear in the evaluated polynomial,
- opening proofs are $\mathbb{F}$-linear in the committed polynomial.

Instantiation of TPC with a KZG Polynomial Commitment.. We now instantiate a linear randomness-free TPC for the special case of a field of primer order $\mathsf{R} = \mathbb{F}$ using a KZG-style polynomial commitment. Let p be a prime and let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ be cyclic groups of order p with a non-degenerate bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ be generators. Fix a maximal degree $\ell > 0$.

- **Setup.**

  $$\mathsf{Gen}(1^\kappa) :$$

  sample $\tau \leftarrow_\$ \mathbb{F}$, and define the SRS

  $$\mathsf{srs} = \big(g_1^{\tau^0}, g_1^{\tau^1}, \ldots, g_1^{\tau^\ell}; g_2^{\tau^0}, g_2^{\tau^1}\big).$$

  Output (srs, τ).
- **Commit.**

  $$\mathsf{Com}^{\mathsf{srs}}(\phi; r) :$$

  let $\phi(X) = \sum_{k=0}^{\ell} a_k X^k \in \mathsf{R}_\ell[X]$. Define

  $$c := g_1^{\phi(\tau)} = \prod_{k=0}^{\ell} \big(g_1^{\tau^k}\big)^{a_k}.$$

  The formal randomness $r \in \mathsf{COIN}^{\mathsf{com}}$ is ignored in this instantiation. Output c.
- **Honest opening.** For $i \in [n]$ and evaluation point $\alpha_i \in \mathbb{F}$,

  $$\mathsf{Open}^{\mathsf{srs}}_{\alpha_i}(\phi; r) :$$

  compute

  $$v_i := \phi(\alpha_i), \qquad q_i(X) := \frac{\phi(X) - v_i}{X - \alpha_i},$$

  and set

  $$\theta_i := g_1^{q_i(\tau)}.$$

  Output (v_i, θ_i), and recall $\mathsf{EOpen}^{\mathsf{srs}}_{\alpha_i}(\phi; r) := (v_i, \theta_i)$.
- **Verification.**

  $$\mathsf{Ver}^{\mathsf{srs}}_{\alpha_i}(c, v_i, \theta_i) :$$

  accept if and only if

  $$e(c/g_1^{v_i},\ g_2) = e(\theta_i,\ g_2^{\tau - \alpha_i}).$$

- **Dummy commitment.**

  $$\mathsf{TCom}^{\tau}(1^\kappa) :$$

  sample $\sigma^\diamond \leftarrow_\$ \mathbb{F}$ and define $c^\diamond := g_1^{\sigma^\diamond}$. Output $(c^\diamond, \sigma^\diamond)$.
- **Simulated opening on J.** For $i \in J \subseteq [n]$ with $|J| \le \ell - 1$,

  $$\mathsf{TOpen}^{\tau}_{\alpha_i}(c^\diamond, \sigma^\diamond) :$$

  Choose $v_i^\diamond \leftarrow_\$ \mathbb{F}$, and define

  $$q_i^\diamond(\tau) := \frac{\sigma^\diamond - v_i^\diamond}{\tau - \alpha_i} \in \mathbb{F}, \qquad \theta_i^\diamond := g_1^{q_i^\diamond(\tau)}.$$

  Output $(v_i^\diamond, \theta_i^\diamond)$. By construction

  $$e(c^\diamond / g_1^{v_i^\diamond}, g_2) = e(g_1, g_2)^{(\sigma^\diamond - v_i^\diamond)} = e(\theta_i^\diamond, g_2^{\tau - \alpha_i}),$$

  so each simulated opening verifies under $\mathsf{Ver}^{\mathsf{srs}}_{\alpha_i}$.
- **Trapdoor collision / explanation.**

  $$\mathsf{TCol}^{\tau}\big(c^\diamond, \sigma^\diamond, \{(\alpha_i, v_i^\diamond, \theta_i^\diamond)\}_{i \in J}, s\big) :$$

  Consider the set of interpolation constraints

  $$\phi^*(0) = s, \quad \phi^*(\tau) = \sigma^\diamond, \quad \phi^*(\alpha_i) = v_i^\diamond \text{ for all } i \in J,$$

  over unknown $\phi^* \in \mathsf{R}_\ell[X]$. Since $|J| \le \ell - 1$, there exists a unique polynomial ϕ^* of degree at most ℓ satisfying all these constraints.

Set $r^* := 0$ and define

$$c^* := \mathsf{Com}^{\mathsf{srs}}(\phi^*; r^*) = g_1^{\phi^*(\tau)} = g_1^{\sigma^\diamond} = c^\diamond.$$

For each $i \in J$, define

$$q_i^*(X) := \frac{\phi^*(X) - v_i^\diamond}{X - \alpha_i}, \qquad \theta_i^* := g_1^{q_i^*(\tau)}.$$

Because $\phi^*(\tau) = \sigma^\diamond$ and $\phi^*(\alpha_i) = v_i^\diamond$, we have

$$q_i^*(\tau) = \frac{\sigma^\diamond - v_i^\diamond}{\tau - \alpha_i},$$

and thus $\theta_i^* = \theta_i^\diamond$ by construction of $\theta_i^\diamond$. Therefore, for all $i \in J$,

$$(v_i^\diamond, \theta_i^\diamond) = \mathsf{EOpen}_{\alpha_i}^{\mathsf{srs}}(\phi^*; r^*).$$

Output (ϕ^*, r^*).

- **Mixed-mode**.

$$\mathsf{TSim}^\tau(\phi, *) :$$

return $\sigma^\diamond = \phi(\tau)$

Equivocation Constraint and Privacy Threshold.. In a classical degree-ℓ Shamir secret-sharing scheme, any $f \le \ell$ corrupted parties learn no information about the secret, while any $\ell + 1$ honest parties can reconstruct it. In our UC-secure realization using the equivocal trapdoor KZG-like polynomial commitment, the simulator must interpolate a polynomial consistent with: (i) the committed secret $\phi(0) = s$, (ii) all dummy openings revealed to the adversary, *and* (iii) the committed value at the trapdoor point $\phi(\tau)$. Thus the retrofitted polynomial must satisfy $|J| + 2$ interpolation constraints, where $J \subseteq [n]$ denotes the set of corrupted parties for which openings have been simulated. Since $\deg(\phi) \le \ell$, we require $|J| \le \ell - 1$. Consequently, the equivocal TPC construction guarantees $(\ell - 1)$-privacy while preserving $\ell + 1$-reconstructability. This slight degradation of the privacy threshold is harmless in our optimized dual-threshold setting: we set $\ell = n - t - 1$ so that privacy holds against any $f < \ell$, while the linear-communication multiplication protocol remains secure for any $f \le t_{\mathrm{acc}} < \ell$, as required.

Degree-0 Case: Classical Equivocal Commitments.. Consider the special case $\ell = 0$, where each commitment is to a constant polynomial $\phi(X) \equiv s \in \mathsf{R}$. Formally, $J = \emptyset$, which means we can ignore the dummy openings in the equivocality property. The TPC reduces exactly to a classical trapdoor commitment interface in the sense of [57] This recovers as a direct instantiation the Pedersen trapdoor commitment scheme in the presentation of Fujisaki [57].

Its algorithms are defined as follows:

- $\mathsf{Gen}(1^\kappa)$: sample $\tau \leftarrow_\$ \mathbb{F}$, compute $\mathsf{srs} := (g, h) = (g, g^\tau)$, and return (srs, τ).
- $\mathsf{Com}^{\mathsf{srs}}(s, r)$: to commit to $s \in \mathsf{R}$ with $r \leftarrow_\$ \mathbb{F}$, and output

$$\mathsf{Com}^{\mathsf{srs}}(s, r) = g^r \cdot h^s.$$

- $\mathsf{Open}^{\mathsf{srs}}(s, r)$: Output (s, r)
- $\mathsf{Ver}^{\mathsf{srs}}(c, s, r)$: Accept if and only if $c = \mathsf{Com}^{\mathsf{srs}}(s, r)$
- $\mathsf{TCom}^\tau(1^\kappa)$: sample $\sigma^\diamond \leftarrow \mathbb{Z}_q$, compute $c^\diamond := g^{\sigma^\diamond}$ and output $(c^\diamond, \sigma^\diamond)$.
- $\mathsf{TCol}^\tau(\sigma^\diamond, s^*)$: on input $s^* \in \mathsf{R}$, output

$$r^* = \sigma^\diamond - s^* \cdot \tau \pmod q.$$

Note that in this case

$$c^\diamond = g^{r^*} \cdot h^{s^*}.$$

- $\mathsf{TSim}^\tau(s, r)$: Output $r + s \cdot \tau \pmod q$

Appendix H.
Universal Composability

We analyze security in the Universal Composability (UC) framework [67], a standard simulation-based model that provides strong, modular composability guarantees.

UC represents distributed systems as collections of (probabilistic) Interactive Machines (IMs). Three types of IM systems are distinguished: (i) *protocol systems*, modeling real protocols and ideal functionalities; (ii) *adversarial systems*, modeling adversaries and simulators; and (iii) *environmental systems*, modeling higher-level protocols that supply inputs, observe outputs, and act as distinguishers. Pure non-determinism, such as message scheduling, is resolved by a master IM, typically under the control of the environment. In practice, however, most meaningful scheduling choices are delegated by the environment to the adversary. For any fixed master IM, the resulting global system induces a probability distribution over executions. The runtime of the overall system, as well as of all its subsystems, is required to be polynomially bounded in the security parameter κ and the size of external inputs, except with negligible probability.

A concrete protocol system $\mathcal{P}$ is said to *emulate* (or *realize*) an abstract protocol system $\mathcal{F}$ if for every admissible adversary $\mathcal{A}$ interacting with $\mathcal{P}$, there exists an admissible adversary $\mathcal{S}$, called a simulator, such that the joint systems $\mathcal{A} \mid \mathcal{P}$ and $\mathcal{S} \mid \mathcal{F}$ are computationally indistinguishable. Formally, for every environment $\mathcal{E}$ interacting with both systems, the probability that $\mathcal{E}$ outputs different decision bits in $\mathcal{E} \mid \mathcal{A} \mid \mathcal{P}$ and $\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$ is negligible in κ, where the probability is taken over the random tapes of all IMs.

A key feature of UC emulation is its composability: if a protocol $\mathcal{P}$ emulates an ideal functionality $\mathcal{F}$, then $\mathcal{P}$ can safely replace $\mathcal{F}$ as a subroutine within any larger protocol without affecting security. In particular, a protocol may assume access to auxiliary ideal functionalities (e.g., $\mathcal{F}_{\text{rbc}}^{\text{acc}}$ or $\mathcal{F}_{\text{idx-acs}}^{\text{acc}}$), which can later be instantiated by their concrete realizations.

Indistinguishability in UC is typically established via a probabilistic *forward* simulation (without rewinding) from $\mathcal{E} \mid \mathcal{A} \mid \mathcal{P}$ and $\mathcal{E} \mid \mathcal{S} \mid \mathcal{F}$. Such simulations guarantee preservation of *hypersafety* properties, meaning that no bad event can occur in the real system unless it is also possible in the ideal one. However, they do not automatically ensure preservation of *(hyper)liveness* properties, i.e., that desired events eventually occur.

To address this gap, we additionally require that the simulation preserves *live executions*, restricted here to *fair executions* [66], [68], [69], in which any action that remains continuously enabled (such as the delivery of a pending message) is eventually triggered within a polynomial number of steps. This notion follows classical liveness-preserving refinement based on fair-trace inclusion [70]. In our setting, this provides a concrete interpretation of liveness at both levels: at the abstract level, fairness ensures that specified post-conditions (e.g., delivery of delayed outputs) eventually occur once their pre-conditions are satisfied; at the fully concrete message-passing level, fair executions include those in which all messages between currently honest parties are eventually delivered (as in [71]). We require that every fair execution of the concrete system $\mathcal{A} \mid \bar{\mathcal{P}}$ is mapped by the simulator to a fair execution of the ideal system $\mathcal{S} \mid \mathcal{P}$, yielding *liveness-preserving emulation*.

Finally, following Shoup et al. [29], [72], [73], [74], [75], we restrict attention to *efficient* protocols, in which the total computation and communication performed by honest parties are bounded by fixed polynomials (independent of the adversary) in the security parameter and the number of parties, except with negligible probability. This efficiency notion is itself composable and rules out degenerate constructions that would trivially satisfy liveness by overwhelming the system, for example by generating a super-polynomial number of events that the master IM cannot process within polynomial time.

Appendix I.
Basic functionalities

This section presents the lower-level functionalities used in our different constructions. As easily accountable agreement functionalities, $\mathcal{F}_{\text{rbc}}^{\text{acc}}$, $\mathcal{F}_{\text{ack}}^{\text{acc}}$, $\mathcal{F}_{\text{idx-acs}}^{\text{acc}}$, $\mathcal{F}_{\text{zkpok}}^{1:\text{M,acc}}$ can be obtained from their non-accountable counterparts $\mathcal{F}_{\text{rbc}}$, $\mathcal{F}_{\text{ack}}$, $\mathcal{F}_{\text{idx-acs}}$, $\mathcal{F}_{\text{zkpok}}^{1:\text{M}}$ by applying the $\mathcal{ABC}$ transformation [26], [28], with additive $O(1)$ round complexity and $O(\kappa n^2)$ communication complexity overhead.

zkPoK An argument of knowledge enables a prover to convince a verifier that he knows a witness for a true NP-statement. If in addition, this proof does not reveal anything beyond the statement, this a zero-knowledge proof of knowledge (ZKPoK). The corresponding $\mathcal{F}_{\text{zkPoK}}$ functionality is presented in Functionality 7. In our construction, all NP-relations are algebraic and express homomorphism relations between abelian groups (e.g., linear relations over $\mathbb{F}_p$ or group-compatible commitments/encryptions). This structure enables the use of efficient Ω-protocols tailored to such linear relations [46], [47].

Reliable Broadcast The accountable reliable broadcast functionality $\mathcal{F}_{\text{rbc}}^{\text{acc}}$ is presented in Functionality 8.

This ensures the following properties:

- **Obligation**: If $f \leq t$, P is correct and triggers $(\mathsf{rb_broadcast}, sid, x)$, then every correct process eventually triggers $(\mathsf{rb_deliver}, sid, P, x)$

Functionality 7 $\mathcal{F}_{\text{zkPoK}}$

Parameters:
$\mathcal{R}$: A NP-relation for an NP language.

Participants:
P: Prover.
V: Verifier.
Sim: Ideal adversary (simulator).

Events:
request (prove, sid, V, x, w): Proving one knows a witness w for x.
notification (verification, sid, P, x): Convinced that P knows a witness for x.

Functionality:
Upon receiving (prove, sid, V, x, w) from P with $\mathcal{R}(x, w) = 1$:
send (verification, sid, P, x) to Sim and V.

Functionality 8 $\mathcal{F}_{\text{rbc}}^{\text{acc}}$

Parameters:
$n, t, f \in \mathbb{N}$, $t < n/3$, $f \in [n]$

Events:
request (rb_bcast, sid, m): The source wishes to reliable-broadcast the message m.
notification (rb_deliver, sid, P, m): The recipient reliable-delivers the message m.

Functionality:
Upon receiving (rb_bcast, sid, m) from the source P for the first time:
Send (rb_deliver, sid, P, m) to Sim and to each $p_j \in \Psi$ as a delayed output

// degraded mode ($f > t$)
Upon receiving (BreakLiveness) from Sim with $f > t$:
Sim can indefinitely delay any output (except detection).

Upon receiving (BreakAgreement, sid, $P, \boldsymbol{\pi}$) from Sim with $f > t$ and P corrupted.
If $\forall i \in [n]$, Judge($\boldsymbol{\pi}[i]$) returns the verdict dis(D_i) with $|D_i| \geq n - 2t$:
send (detection, sid_P, D_i, $\boldsymbol{\pi}[i]$) to each $p_i \in \Psi$ as a delayed output.
allow conflicting notifications of the form
(rb_deliver, sid, P, x) and (rb_deliver, sid, $P, x' \neq x$)

- **Justification**: If a correct process triggers (rb_deliver, sid, P, x), with P correct, then P has triggered (rb_broadcast, sid, P, x)
- **Agreement**: If $f \leq t$, then no two correct processes trigger (rb_deliver, sid, P, x) and (rb_deliver, $sid, P, x' \neq x$)
- **Totality**: If $f \leq t$ and a correct process triggers (rb_deliver, sid, P, x), then every correct process eventually (rb_deliver, $sid, P, *$)
- **Accountability w.r.t. Agreement**: If two correct processes trigger notifications (rb_deliver, sid, P, x), and (rb_deliver, $sid, P, x' \neq x$) respectively, then every correct process eventually receives an externally-verifiable proof-of-misbehavior against $n - 2t$ dishonest processes.

$\mathcal{F}_{\text{rbc}}$ can be realized with $O(|m|n + n^2 \log n)$ communication and $O(1)$ rounds [76].

Acknowledgement Functionality $\mathcal{F}_{\text{ack}}$ (see Functionality 9) allows processes to acknowledge a statement promoted by a given dealer. The statement can be a session ID, allowing processes to acknowledge the successful completion of the associated task driven by the dealer. If an honest process receives an acknowledgment, then all honest processes will eventually receive the same acknowledgment.

Functionality $\mathcal{F}_{\text{ack}}$ can be trivially realized on top of $\mathcal{F}_{\text{rbc}}$ and $(n - t)$-threshold signatures (see Algorithm 10). Here the threshold signature can be implemented by the trivial concatenation of simple digital signatures, while maintaining a communication complexity of $O(n|x| + \kappa n^2)$.

One-to-many zkPoK functionality $\mathcal{F}_{\text{zkPoK}}^{1:\text{M,acc}}$**.** The functionality $\mathcal{F}_{\text{zkPoK}}^{1:\text{M,acc}}$ (see Functionality 10) combines properties of $\mathcal{F}_{\text{zkPoK}}$ and $\mathcal{F}_{\text{rbc}}^{\text{acc}}$. Here, a prover aims to convince verifiers (i.e., all other processes) that he "knows" a witness w for a true NP-statement x, without revealing any additional information. In addition, of the $\mathcal{F}_{\text{zkpok}}$'s properties of completeness, soundness, zero-knowledge, and knowledge extractability, for a specific session, $\mathcal{F}_{\text{zkPoK}}^{1:\text{M,acc}}$ guarantees the properties of accountable reliable broadcast:

- **Agreement & Totality**: If $f \leq t$ and a correct verifier is convinced by a statement x, every correct verifier is eventually convinced by the same statement x.
- **Accountability**: If a disagreement occurs and $f \leq t_{acc}$, every judge instance associated with an honest party eventually returns a verdict exposing a significant set of dishonest parties.

Functionality $\mathcal{F}_{\text{zkpok}}^{1:\text{M}}$ can be trivially realized (see Algorithm 11) by composing $\mathcal{F}_{\text{zkpok}}$ with $\mathcal{F}_{\text{ack}}$.

Functionality 9 $\mathcal{F}_{\text{ack}}$ – Acknowledgement Gate

Parameters:
$n, t, f \in \mathbb{N}$, $t < n/3$, $f \in [n]$; designated dealer $D \in \Psi$;

Variables:
$\mathsf{A}[\mathsf{sid}][x] \subseteq \Psi$ (Initially $\mathsf{A}[\mathsf{sid}][x] = \emptyset$) ▷ parties who *approved* statement x

Interface (events):
request $(\mathsf{Approve}, \mathsf{sid}, x)$ ▷ if $x = \mathsf{sid}$, x is omitted
notification $(\mathsf{Ack}, \mathsf{sid}, x)$

Functionality:
Approval collection
Upon $(\mathsf{Approve}, \mathsf{sid}, x)$ from $p \in \Psi$:
  $\mathsf{A}[\mathsf{sid}][x] \leftarrow \mathsf{A}[\mathsf{sid}][x] \cup \{p\}$; send $(\mathsf{approve}, \mathsf{sid}, x, p)$ to Sim

Acknowledge rule
Upon $D \in \mathsf{A}[\mathsf{sid}][x]$ **and** $|\mathsf{A}[\mathsf{sid}][x]| \geq n-t$: ▷ Obligation and Justification
  Send $(\mathsf{Ack}, \mathsf{sid}, x)$ to *all* parties as delayed outputs. ▷ Totality

// degraded mode ($f > t$)
Upon receiving $(\mathsf{BreakLiveness})$ from Sim with $f > t$:
  Sim can indefinitely delay any output.

Algorithm 10 $\mathcal{P}_{\text{ack}} \leq \mathcal{F}_{\text{acc}}$ via $(\mathcal{F}_{\text{rbc}}, \mathcal{F}_{\text{tss}})$: Pseudocode (for process p_i)

Uses:
  $\mathcal{F}_{\text{rbc}}$ ▷ reliable broadcast
  $\mathcal{F}_{\text{tss}}$ ▷ threshold signature scheme

Parameters:
  L: leader

upon request $(\mathsf{approve}, \mathsf{sid}, x)$:
  send $(\mathsf{share_sign}, \mathsf{sid}, \langle \text{ACK}, x \rangle)$ to $\mathcal{F}_{\text{tss}}$

upon receiving $(\mathsf{share_signature}, \mathsf{sid}, \sigma_i, \langle \text{ACK}, x \rangle)$ from $\mathcal{F}_{\text{tss}}$:
  send $(\mathsf{send}, \mathsf{sid}, \sigma_i, \langle \text{ACK}, x \rangle)$ to L

upon receiving $(\mathsf{rcv}, \mathsf{sid}, \sigma_j, \langle \text{ACK}, x \rangle)$ from p_j:
  L **sends** $(\mathsf{share_verify}, \mathsf{sid}, p_j, \sigma_j, \langle \text{ACK}, x \rangle)$ to $\mathcal{F}_{\text{tss}}$

upon receiving $(\mathsf{share_verification}, \mathsf{sid}, p_j, \sigma_j, \langle \text{ACK}, x \rangle, \mathit{true})$ from $\mathcal{F}_{\text{tss}}$:
  $S \leftarrow S \cup \{(\sigma_j, \langle \text{ACK}, x \rangle)\}$

upon $|S| = n - t$:
  send $(\mathsf{combine}, \mathsf{sid}, S)$ to $\mathcal{F}_{\text{tss}}$

upon receiving $(\mathsf{th_signature}, \mathsf{sid}, \sigma^T, \langle \text{ACK}, x \rangle)$ from $\mathcal{F}_{\text{tss}}$:
  send $(\mathsf{rb_bcast}, \mathsf{sid}, (\sigma^T, \langle \text{ACK}, x \rangle))$ to $\mathcal{F}_{\text{rbc}}$

upon receiving $(\mathsf{rb_deliver}, \mathsf{sid}, P, (\sigma^T, \langle \text{ACK}, x \rangle))$ from $\mathcal{F}_{\text{rbc}}$:
  send $(\mathsf{th_verify}, \mathsf{sid}, \sigma^T, \langle \text{ACK}, x \rangle)$ to $\mathcal{F}_{\text{tss}}$

upon receiving $(\mathsf{th_verification}, \mathsf{sid}, \sigma^T, \langle \text{ACK}, x \rangle, \mathit{true})$ from $\mathcal{F}_{\text{tss}}$:
  trigger $(\mathsf{approve}, \mathsf{sid}, P, x)$

Index Agreement on a Core Set functionality $\mathcal{F}^{\text{acc}}_{\text{idx_acs}}$ In the context of index ACS, *process validation* refers to the mechanism by which a process considers that another process has done some action correctly (e.g., complete secret sharing). Each process p_i maintains a dynamic set of processes it has validated (in our case, from which it has received its secret share). This set can grow over time but never decrease (a process cannot be "un"-validated). A set of processes V is said *locally validated* (resp., *globally validated*) if every process $p_j \in V$ has been validated by at least one correct process (resp. by all the correct processes). Let us note that if V is a singleton, it boils down to the definition from [31]. An essential property of the validated set in index ACS is *Totality*. This property ensures that if a set of processes is locally validated at some time, and $f \leq t$, then it will be eventually globally validated. In practice, this property is (typically) ensured by the Totality/Completeness property of ACSS. The functionality ensures the following properties:

- **Obligation**: If $f \leq t$, and $\exists Q \subseteq [n]$, $|Q| \geq n-t$, $\forall j \in Q$: $(\mathsf{validate}, \mathsf{sid}, j)$ has been triggered by all correct processes, then every correct process eventually triggers $(\mathsf{decide}, sid, *)$
- **Validity**: If a correct process p_i triggers $(\mathsf{decide}, sid, Q)$, then Q counts (at least) $n - t$ process IDs validated by p_i
- **Agreement**: If $f \leq t$, no pair of correct processes trigger $(\mathsf{decide}, sid, Q)$ and $(\mathsf{decide}, sid, Q' \neq Q)$
- **Accountability w.r.t. Agreement**: If two correct processes trigger notifications $(\mathsf{decide}, sid, Q)$, and $(\mathsf{decide}, sid, Q' \neq Q)$ respectively, then every correct process eventually receives an externally-verifiable proof-of-misbehavior against $n - 2t$ dishonest processes.

Functionality 10 $\mathcal{F}_{\text{zkPoK}}^{1:\text{M},\text{acc}}$

Parameters:
$\mathcal{R}$: An NP relation for an NP language.
$n, t, f, t_{snd} \in \mathbb{N}$, $t < n/3$, $f \in [n]$, $t_{snd} \in [n]$ ▷ $t_{snd} \geq n - t > t_{acc}$ requires $\mathcal{F}_{\text{nizkPoK}}$

Events:
request (prove, sid, x, w): Proving knowledge of a witness for the statement x.
notification (verification, sid, P, x): Verifying that the prover P knows a witness for x.

Functionality:
// nominal mode ($f \leq t$)
Upon receiving (prove, sid, x, w) from P for the first time for sid, s.t. $(x, w) \in \mathcal{R}$:
send (verification, sid, P, x) directly to Sim and to each $p_j \in \Psi$ as a delayed output.
// degraded mode ($f > t$)
Upon receiving (BreakLiveness) from Sim with $f > t$:
Sim can indefinitely delay any output (except detection).
Upon receiving (BreakAgreement, sid, $P, \boldsymbol{\pi}$) from Sim with $f > t$ and P corrupted.
If $\forall i \in [n]$, Judge($\boldsymbol{\pi}[i]$) returns the verdict dis(D_i) with $|D_i| \geq n - 2t$:
send (detection, sid_P, $D_i, \boldsymbol{\pi}[i]$) to each $p_i \in \Psi$ as a delayed output.
allow conflicting, though still sound, notifications of the form
(verification, sid, idx, P, x) and (verification, sid, $idx, P, x' \neq x$)

// (uncovered) catastrophic mode ($f > t_{snd}$)
Upon receiving (BreakSoundness, sid, P) from Sim with $f > t_{snd}$ and P corrupted.
allow any (non-sound) statement notification of the form
(verification, sid, idx, P, x), without requiring a witness w, s.t. $(x, w) \in \mathcal{R}$.

Algorithm 11 $\mathcal{P}_{\text{zkPoK}}^{1:\text{M}} \leq \mathcal{F}_{\text{zkPoK}}^{1:\text{M}}$ via $(\mathcal{F}_{\text{zkPoK}}, \mathcal{F}_{\text{ack}})$: Pseudocode (for process p_i)

Uses:
    $\mathcal{F}_{\text{zkPoK}}$ parametrized with an NP relation $\mathcal{R}$ ▷ possibly interactive
    $\mathcal{F}_{\text{ack}}$ ▷ acknowledgement

Parameters:
    P: prover

upon requesting (prove, sid, x, w):
    for each $p_j \in \Psi$:
        send (prove, sid, ssid_j, p_j, x, w) to $\mathcal{F}_{\text{zkPoK}}$

upon receiving (proof, sid, ssid_j, P, x) from $\mathcal{F}_{\text{zkPoK}}$:
    send (approve, sid, x) to $\mathcal{F}_{\text{ack}}$

upon receiving (ack, sid, x) from $\mathcal{F}_{\text{ack}}$:
    trigger (verification, sid, P, x)

The accountable index agreement on a core set functionality $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$ is presented in Functionality 11. $\mathcal{F}_{\text{idx_acs}}$ can be realized with $O(\kappa n^3)$ communication complexity and $O(1)$ round complexity, assuming hash functions with collision resistance and linear hiding [31], [74].

Functionality 11 $\mathcal{F}_{\text{idx_acs}}^{\text{acc}}$ – Accountable Index Agreement on a Core Set

Parameters:
$n, t, f \in \mathbb{N}$, $t < n/3$, $f \in [n]$

Variables:
$\mathsf{Validated}[\mathsf{sid}][i], \mathsf{Validators}[\mathsf{sid}][i] \subseteq [n]$, initially $\emptyset$, for each sid and $p_i \in \Psi$.

Events / Interface:
request $(\mathsf{validate}, \mathsf{sid}, j)$ from p_i
notification $(\mathsf{decide}, \mathsf{sid}, Q)$ to p_i
notification $(\mathsf{validation}, \mathsf{sid}, i, j)$ to Sim
request $(\mathsf{BreakAgreement}, \mathsf{sid}, \pi)$ from Sim
request $(\mathsf{BreakLiveness}, \mathsf{sid})$ from Sim

Functionality:
Upon receiving $(\mathsf{validate}, \mathsf{sid}, j)$ from p_i:
$\mathsf{Validated}[\mathsf{sid}][i] \leftarrow \mathsf{Validated}[\mathsf{sid}][i] \cup \{j\}$
$\mathsf{Validators}[\mathsf{sid}][j] \leftarrow \mathsf{Validators}[\mathsf{sid}][j] \cup \{i\}$
$\mathfrak{Q}[\mathsf{sid}] = \{Q \subseteq [n] \mid |Q| \geq n - t, \forall j \in Q, |\mathsf{Validators}[\mathsf{sid}][j]| \geq t + 1\}$
Forward $(\mathsf{validation}, \mathsf{sid}, i, j)$ to Sim

Upon receiving $(\mathsf{chooseSet}, \mathsf{sid}, Q \in \mathfrak{Q}[\mathsf{sid}])$ from Sim:
$\forall i \in [n]$: send $(\mathsf{decide}, \mathsf{sid}, Q)$ to p_i as a delayed output when $Q \subseteq \mathsf{Validated}[\mathsf{sid}][i]$

Upon $\exists Q \in \mathfrak{Q}[\mathsf{sid}]$, s.t. $\forall j \in Q$, $\mathsf{Correct} \subseteq \mathsf{Validators}[\mathsf{sid}][j]$:
Order Sim to provide a $(\mathsf{chooseSet}, \mathsf{sid}, Q' \in \mathfrak{Q}[\mathsf{sid}])$ request.

// degraded mode ($f > t$)
Upon receiving $(\mathsf{BreakLiveness})$ from Sim with $f > t$:
Sim can indefinitely delay any output (except detection).

Upon receiving $(\mathsf{BreakAgreement}, \mathsf{sid}, P, \boldsymbol{\pi})$ from Sim with $f > t$.
If $\forall i \in [n]$, $\mathsf{Judge}(\boldsymbol{\pi}[i])$ returns the verdict $\mathsf{dis}(D_i)$ with $|D_i| \geq n - 2t$:
send $(\mathsf{detection}, \mathsf{sid}, D_i, \boldsymbol{\pi}[i])$ to each $p_i \in \Psi$ as a delayed output.
allow conflicting, though still valid, notifications of the form
$(\mathsf{decide}, \mathsf{sid}, Q)$ and $(\mathsf{decide}, \mathsf{sid}, Q' \neq Q)$

Appendix J.
The public judge and the judicial functionality

We specify the public judge Judge (Global Functionality 14) and the auxiliary judicial functionality $\mathcal{F}_{\text{judi}}$ (Functionality 12–13); their parallel composition yields the globally available functionality $\mathcal{J} \triangleq \mathsf{Judge} \mid \mathcal{F}_{\text{judi}}$. The functionality $\mathcal{F}_{\text{judi}}$ is the multi-signature analog of the certification functionality $\mathcal{F}_{\text{cert}}$ and is realized from a (multi-)signature functionality $\mathcal{F}_{(\text{m})\text{sig}}$ together with a certification authority (PKI) $\mathcal{F}_{\text{CA}}$, following [65].

On a request $\big(\mathsf{verdict}, \mathsf{sid}, (m^1, Q^1, \sigma^1), (m^2, Q^2, \sigma^2)\big)$, Judge queries $\mathcal{F}_{\text{judi}}$ to verify that, for each $i \in \{1, 2\}$, σ^i is a valid multi-signature by Q^i on m^i; if both verify and the messages conflict (e.g., same header, distinct payload), it returns the verdict $\mathsf{dis}(Q^1 \cap Q^2)$. Making Judge globally accessible captures *external verifiability*: third parties that did not participate in the execution can validate evidence.

The same interface is used to validate BreakAgreement requests: before an accountable functionality allows an agreement break, it checks that the attached proof (obtainable by the ideal adversary via $\mathcal{F}_{\text{judi}}$) incriminates at least $n - 2t$ processes. In our construction, accountability reduces to the accountable agreement of the confirmer $\mathcal{F}_{\text{conf}}^{\text{acc}}$; intermediate layers merely forward proofs of misbehavior (and derived verdicts) upward. We omit evidence-collection details at those layers, as they arise from the concrete realization of $\mathcal{F}_{\text{conf}}^{\text{acc}}$ in [28].

Functionality 12 $\mathcal{F}_{\mathsf{judi}}$ – Signing, Aggregating

//A certification authority functionality $\mathcal{F}_{\mathrm{ca}}$ allows processes to agree on one unique public key per process

Signature Generation:

Upon receiving $(\mathsf{sign}, \mathsf{sid}, p_i, m)$ from p_i:
- Forward $(\mathsf{sign}, \mathsf{sid}, p_i, m)$ to the adversary.
- Upon receiving $(\mathsf{signature}, \mathsf{sid}, p_i, m, \sigma_i)$ from the adversary:
  - If no entry $(\mathsf{sid}, m, \sigma_i, 0)$ is recorded:
    - Record the entry $(\mathsf{sid}, m, \sigma_i, \{p_i\}, 1)$.
    - Output $(\mathsf{signature}, \mathsf{sid}, p_i, m, \sigma_i)$ to p_i.

Signature Combination:

Upon receiving $(\mathsf{cb}, \mathsf{sid}, G, m, \{\sigma_j\}_{p_j \in G})$, from p_i, s.t. $G \subseteq \Psi \wedge |G| \geq 2$:
- Forward $(\mathsf{cb}, \mathsf{sid}, G, m, \{\sigma_j\}_{p_j \in G})$ to the adversary.
- Upon receiving $(\mathsf{cb}, \mathsf{sid}, G, m, \{(\phi_j)\}_{p_j \in G}, \sigma_G)$ from the adversary:
  - Determine $f_G \leftarrow \mathsf{BatchIndividualVerif}(\mathsf{sid}, G, m, \{(\sigma_j, \phi_j)\}_{p_j \in G})$
  - If $f_G = 1$:
    - Record the entry $(\mathsf{sid}, m, \sigma_G, G, 1)$.
    - Output $(\mathsf{multisignature}, \mathsf{sid}, G, m, \sigma_G)$ to p_i.
  - Else:
    - Output $(\mathsf{multisignature}, \mathsf{sid}, G, m, \perp)$ to p_i.

Functionality 13 $\mathcal{F}_{\mathsf{judi}}$ – Verifying

Signature Verification:

Upon receiving $(\mathsf{verify}, \mathsf{sid}, G, m, \sigma_G)$, s.t. $G \subseteq \Psi \wedge |G| \geq 1$:
- Forward $(\mathsf{verify}, \mathsf{sid}, G, m, \sigma_G)$ to the adversary.
- Upon receiving $(\mathsf{verified}, \mathsf{sid}, G, m, \phi_G)$ from the adversary:
  - Determine $f_G \leftarrow \mathsf{Verif}(\mathsf{sid}, G, m, \sigma_G, \phi_G)$
  - Output $(\mathsf{verified}, \mathsf{sid}, G, m, \sigma_G, f_G)$ to the requester (process or judge).

Verification Procedure:

$\mathsf{Verif}(\mathsf{sid}, G, m, \sigma, \phi_G)$:
- **(1)** If the entry $(\mathsf{sid}, m, \sigma, G, 1)$ is recorded: return $f = 1$.
  (Completeness: Verification succeeds for valid (multi-)signatures.)
- **(2)** Else, if $\exists p_j \in G \cap \mathsf{Correct}$, and no entry $(\mathsf{sid}, m, \sigma', G' \ni p_i, 1)$ is recorded:
  record the entry $(\mathsf{sid}, m, \sigma, G, 0)$ and return $f = 0$.
  (Unforgeability: Verification fails if one signer did not sign m.)
- **(3)** Else, if an entry $(\mathsf{sid}, m, \sigma, G, f')$ is recorded: return $f = f'$.
  (Consistency: Identical parameters produce the same verification result.)
- **(4)** Else: record the entry $(\mathsf{sid}, m, \sigma, G, \phi_G)$ and return $f = \phi_G$.

$\mathsf{BatchIndividualVerif}(\mathsf{sid}, G, m, \{(\sigma_j, \phi_j)\}_{p_j \in G})$:
- Determine $\{f_j \leftarrow \mathsf{Verif}(\mathsf{sid}, \{p_j\}, m, \sigma_j, \phi_j)\}_{p_j \in G}$
- Return $\bigwedge_{p_j \in G} f_j$

Functionality 14 Judge – Globally available

Verdict request:

Upon receiving $\big(\mathsf{verdict}, \mathsf{sid}, (m^1, Q^1, \sigma^1), (m^2, Q^2, \sigma^2)\big)$ from a requester R:
- $\forall i \in \{1, 2\}$: send $(\mathsf{verify}, \mathsf{sid}, Q^i, m^i, \sigma^i)$ to $\mathcal{F}_{\mathrm{judi}}$
- When $(\mathsf{verified}, \mathsf{sid}, Q^i, m^i, \sigma^i, f^i)$ is received from $\mathcal{F}_{\mathrm{judi}}$ for each $i \in \{1, 2\}$:
  - **if** $m^1.\mathsf{header} = m^2.\mathsf{header} \wedge m^1.\mathsf{payload} \neq m^2.\mathsf{payload} \wedge f^1 = f^2 = true$:
    send $\big(\mathsf{verdict}, \mathsf{sid}, \mathsf{dis}(Q^1 \cap Q^2)\big)$ to R. **(equivocation detected)**
  - **else**
    send $(\mathsf{verdict}, \mathsf{sid}, \perp)$ to R. **(insufficient evidence)**