AsyRand: An Asynchronous Randomness Beacon via Look-Ahead Agreement

Liang Zhang*, Binyang Hou†, Zhengxuan Guo*, Haibin Kan†, Jiheng Zhang*, and Moti Yung‡
*The Hong Kong University of Science and Technology
†Fudan University
‡Google LLC and Columbia University

***Abstract*—Asynchronous randomness beacons must preserve agreement and sustain output generation despite Byzantine faults while preventing adversarial bias. We present AsyRand, a publicly verifiable randomness beacon that combines publicly verifiable secret sharing (PVSS) with local-coin asynchronous binary agreement (ABA). Its Producer-Consumer design overlaps PVSS preparation with beacon generation and consumes one accepted PVSS object per output. Column agreement amortizes coordination across multiple outputs. Its look-ahead agreement fixes the successor column's accepted contributions before current contributions are revealed, while recent-contributor exclusion ensures an honest contribution in every $f+1$ consecutive outputs, where f bounds the number of Byzantine parties. Under a production window ≥ 2, every fixed epoch completes with probability one. These mechanisms, together with PVSS secrecy and beacon chaining, provide unpredictability and bias resistance for fixed targets at least $f+1$ epochs ahead. The communication complexity is $O(n^2)$ per output in the optimistic case. We instantiate AsyRand using ALBATROSS PVSS (Cascudo et al., ASIACRYPT'20) and WaterBear Quadratic-ABA (Zhang et al., Usenix Security'23). Experiments with 16–121 parties across eight regions show that AsyRand achieves higher throughput than our implementations of GRandLine (Bacho et al., CCS'24) and SPURT (Das et al., S&P'22). In the fault-free case, at 121 parties, AsyRand achieves $1.66\times$ and $14.03\times$ their respective throughput.**

1. Introduction

A randomness beacon [1], introduced by Rabin over four decades ago, produces a publicly verifiable sequence of random values. A randomness beacon protocol should preserve agreement, public verifiability, bias resistance, and unpredictability [2]. Randomness beacons underpin important applications such as leader and committee election [3], [4], [5], [6], blockchains and smart contracts [6], [7], [8], sharding [9], [10], anonymous communication [11], and secure computation [12].

Secret-sharing-based schemes allow distributed parties to contribute hidden randomness with fault tolerance. Verifiable secret sharing (VSS) [15], [37] and publicly verifiable secret sharing (PVSS) [13], [14], [27], [28], [29], [60] bind a party to a hidden entropy before threshold reconstruction. In a fully asynchronous setting, where messages may be delayed arbitrarily, the Fischer–Lynch–Paterson (FLP) impossibility result rules out deterministic agreement with guaranteed termination [16]. The reason is that unbounded delay makes a faulty party indistinguishable from a slow but honest one. Waiting indefinitely sacrifices progress, whereas proceeding based on local observations can cause honest parties to diverge or allow an adversary to influence which hidden values are released. Consequently, many distributed randomness beacon protocols assume a synchronous or partially asynchronous network [13], [14], [15], [27], [29], [60]. HashRand [37] is an asynchronous beacon protocol that employs approximate agreement [38] to achieve Monte Carlo-style termination: it terminates within a bounded number of rounds while permitting only a small, configurable probability of disagreement.

To overcome the FLP problem in asynchronous environments, randomized protocols have been proposed that terminate with probability one [40]. Common coins [43], [47] are widely used to gain randomization for asynchronous protocols, including asynchronous binary agreement (ABA) [46], [48], [49], [50], multi-valued asynchronous Byzantine agreement (MVBA) [51], [52], [53], asynchronous common subset (ACS) [42], [54] and asynchronous BFT consensus [55], [56], [57]. Distributed key generation (DKG) is commonly used to establish threshold keys for common-coin protocols. Accordingly, asynchronous DKG (ADKG) constructions leverage internally generated coins or randomness to drive agreement [19], [20], [21], [23]. However, a bootstrapping concern remains for asynchronous beacon constructions that rely on an external common coin: such constructions need enough shared randomnesses to coordinate beacon generation, yet a sufficiently strong and continuously available common-coin service would already provide much of the functionality expected from a beacon.

To circumvent this bootstrapping concern and avoid the need of a DKG setup, we propose a fully asynchronous randomness beacon framework AsyRand that combines PVSS [24] with local-coin ABA [58], enabling parties to reach agreement without relying on common coins. PVSS schemes enables parties to commit to hidden contributions and later reconstruct them with public verifiability. Local-coin ABA achieves binary agreement in asynchronous settings using only local generated randomness.

Combining PVSS with local-coin ABA introduces two related requirements. The protocol must accept enough

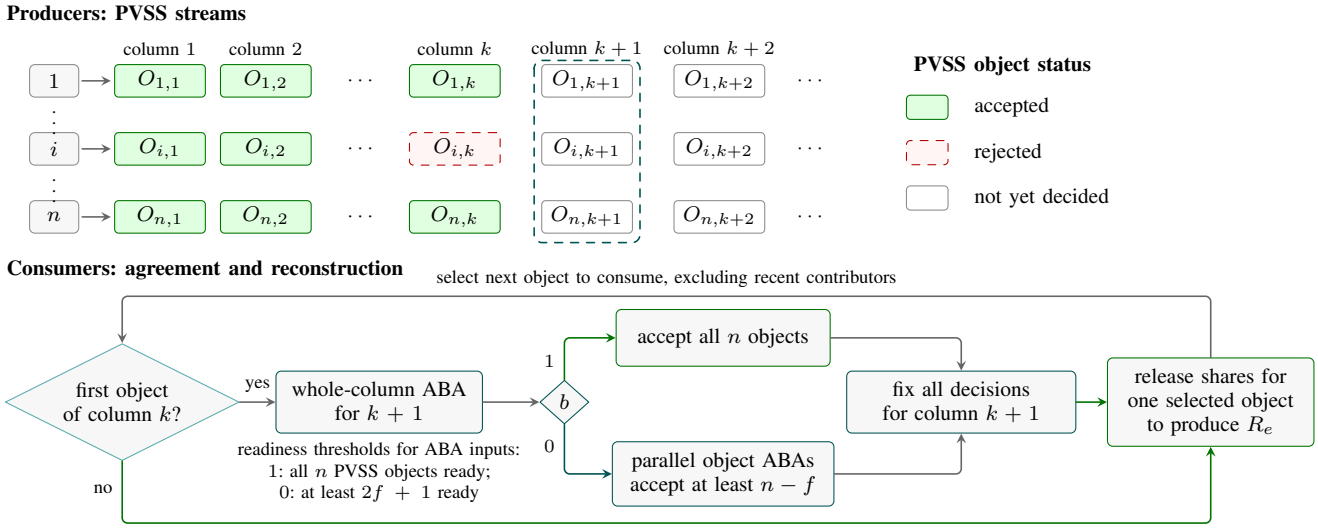

Figure 1. Overview of AsyRand with n parties and at most f faults. Producers generate ordered PVSS streams: $O_{i,k}$ denotes Producer i's object at position k, and equal positions form columns. Consumers fix all decisions and accepted object hashes for column $k+1$ before releasing shares for column k. Whole-column decision $b=1$ accepts all n objects. $b=0$ runs parallel object ABAs, allowing missing-object input 0 only after $n-f$ ABA decision certificates for 1. The first object of each column waits for these decisions. Later objects reuse them and proceed directly to share release. Each accepted object yields one beacon R_e, where e is the epoch.

PVSS objects from different parties to sustain beacon output generation. It must also fix which objects are accepted for future outputs before revealing the current hidden values. Meeting these requirements poses the following interconnected design challenges for AsyRand:

- **Commitment.** Fixing PVSS contributions and their acceptance before revelation.
- **Randomness.** Ensuring unpredictability and bias resistance for future beacon outputs.
- **Consistency.** Preserving agreement and public verifiability despite malicious parties.
- **Liveness.** Guaranteeing output delivery to all honest parties despite malicious parties.
- **Efficiency.** Maintaining computational and communicational efficiency for each beacon output.

We first organize PVSS objects in a distributed Producer-Consumer structure. Each party runs two concurrent processes: a Producer and a Consumer. The Producer prepares an ordered stream of PVSS objects, each containing encrypted shares and proofs for one hidden contribution. Every Producer disseminates its objects via reliable broadcast (RBC). The Consumer agrees on a common set of objects to accept and reconstructs their corresponding values. The Producer-Consumer structure lets Producers supply the next required objects without waiting for beacon output or ABA execution results. A production window of size $W \geq 2$ limits how many unconsumed stream positions each Producer may prepare in advance. Objects at the same stream position form a column, with one position per Producer. Figure 1 illustrates these Producer streams and the Consumer's agreement and reconstruction flow.

To reduce per-output overhead, AsyRand consumes one accepted PVSS object for each beacon output. This avoids the aggregation of multiple parties' PVSS contributions, as used in SPURT [27] and OptRand [14]. To ensure regular honest contributions, we require every sequence of $f+1$ consecutive outputs to include an honest contribution, where at most f Producers are faulty. Selecting distinct Producers within each such sequence guarantees this property [13], [29], motivating the exclusion of recent contributors.

Agreement on individual PVSS objects alone does not ensure enough accepted contributions from distinct Producers. Rejecting objects merely because they have not yet been delivered can exclude delayed honest contributions. Without contributor exclusion, faulty Producers may supply all contributions in a sequence of $f+1$ outputs, leaving no honest contribution to support the security argument. With exclusion, too few accepted Producers may leave no eligible object and halt progress. Consumers must also prevent revealed randomness from influencing future acceptance decisions. We address this requirement through *look-ahead agreement*, which combines column agreement with a *look-ahead barrier* on reconstruction.

For each column, Consumers accept at least $q=n-f$ objects from distinct Producers. Consumers first run a whole-column ABA to decide whether to accept all n objects. A decision of 0 invokes parallel object ABAs to determine individual acceptance. To prevent premature rejection, a Consumer may propose 0 for an undelivered object only after obtaining decision-1 certificates for q other objects in the same column. Fresh local coins allow ABA to operate independently of beacon outputs. The look-ahead barrier delays any honest reconstruction share from the current

column until all successor-column acceptance decisions and the contents of accepted objects are fixed.

At beacon epoch e, Consumers use the previous beacon value R_{e-1} to deterministically select an accepted, unconsumed object from the current column, excluding recent contributors across column boundaries. Parties broadcast reconstruction shares for the selected object. Consumers verify these shares, reconstruct the object's hidden value Z from $f+1$ valid shares from distinct parties, and compute $R_e = H_{\mathsf{beacon}}(e \parallel R_{e-1} \parallel Z)$. Reconstruction remains possible even if the Producer stops responding. Consumers then construct an output record binding R_e to the preceding record, the consumed object, and the column decisions. They mark the object as consumed and update the recent-contributor queue before proceeding to the next output.

Within this design, Producers prepare and buffer PVSS objects in advance, overlapping production with consumption. When production and dissemination keep pace with consumption, beacon throughput is governed primarily by the Consumer path, including amortized column agreement and the reconstruction of one PVSS object per output. The Consumer's PVSS reconstruction work requires $O(t)$ group operations and no pairings per output with ALBATROSS PVSS [26] instantiation, where $t = f+1$. Consequently, AsyRand achieves higher throughput than related works, as demonstrated in Section 8.

Our contributions are as follows:

- **Asynchronous beacon framework.** We present a fully asynchronous beacon framework that combines PVSS contributions with local-coin ABA. Its distributed Producer-Consumer model separates PVSS preparation from beacon generation, enabling continuous output without an external common coin or shared-key DKG. The framework preserves agreement and completes almost surely, providing Las Vegas termination. We instantiate it with ALBATROSS PVSS [26] and WaterBear Quadratic-ABA [58].
- **Look-ahead agreement.** We organize RBC and local-coin ABA into a column-agreement protocol that selects a common set of at least $n-f$ valid PVSS objects, with a whole-column acceptance fast path. This provides a validated asynchronous common subset for beacon generation. We couple this subset selection with a look-ahead barrier that fixes all successor-column acceptance decisions and accepted object contents before any honest reconstruction share from the current column is released.
- **Security guarantees.** We prove safety and public verifiability against static Byzantine adversaries corrupting at most $f < n/3$ parties. Our analysis combines honest-contributor coverage, look-ahead agreement, PVSS IND1-secrecy, and beacon chaining to establish unpredictability and bias resistance for fixed target epochs at least $f+1$ epochs ahead. With correct primitives, eventual honest-to-honest message delivery, continued honest execution, and a production window $W \geq 2$, every fixed epoch completes with probability one.
- **Implementation and experimental evaluation.** We implement AsyRand with pairing-free ALBATROSS PVSS reconstruction, overlapping advance production with consumption and amortizing agreement across column outputs. We evaluate our implementation against GRandLine and SPURT under the same transport and worker-pool configuration. Experiments span 16–121 parties across eight AWS regions. AsyRand achieves higher steady-state throughput at every tested size and sends 51.2–62.4% less data per output than SPURT. At 61 parties, it achieves $2.11\times$ GRandLine's throughput while sending 23.8% less data per output.

2. Related Works

Public randomness can be obtained from proof-of-work (PoW) blockchains [30], [31], verifiable random functions (VRFs) [3], [6], verifiable delay functions (VDFs) [32], [33], and threshold signatures [4], [34]. Choi et al. [2] survey how these approaches distribute trust and impose different timing and setup requirements. We focus on sharing-based constructions and the coordination needed to fix contributions before revelation, tolerate unavailable Producers, and deliver verifiable outputs in a fully asynchronous network. Table 1 summarizes representative beacons and common coins under their respective models.

Sharing-based randomness beacons. Verifiable secret sharing [35], [36] binds a party to a contribution that a threshold set of parties holding shares can recover. PVSS [24], [25] additionally permits public validation of the sharing transcript. HydRand [29] uses PVSS to generate a continuous beacon under synchrony without a trusted dealer. RandPiper [13] combines state-machine replication with PVSS or VSS, exposing tradeoffs between communication, unpredictability distance, and corruption models. SPURT [27] operates under partial synchrony with transparent setup, whereas OptRand [14] provides an optimistically responsive synchronous construction supporting reconfiguration. Rondo [15] combines batched asynchronous sharing with partial output and a partially synchronous BFT protocol. GRandLine [28] establishes keys through a synchronous, adaptively secure DKG and subsequently generates beacon values in asynchronous rounds. These works motivate distinguishing the network assumptions of sharing, initialization, and output coordination.

Asynchronous beacons and termination. RandHerd [60] uses its RandHound for initial grouping and threshold-key establishment, then combines group threshold Schnorr signatures through CoSi [61] to generate publicly verifiable outputs. Its availability requires each group to retain enough honest members to generate its threshold signature. Similarly, GRandLine [28] operates in an asynchronous network after establishing its PVSS-based DKG. Freitas et al. [38] construct approximate and Monte Carlo (MC) common coins from bundled approximate agreement, permitting outputs that are close or allowing a configurable

TABLE 1. COMPARISON WITH DISTRIBUTED RANDOMNESS BEACONS.

Protocol	Network	Coord.	Crypto.	Live.	Comm.	Unpred.	Bias-resist.	Comp.	Verif.	Setup
HydRand [29]	sync.	sync BA	PVSS	D	$O(n^2)$	$f+1$	✓	$O(n)$	$O(n)$	PKI
GRandPiper [13]	sync.	BFT-SMR	PVSS	D	$O(n^2)$	$f+1$	✓	$O(n^2)$	$O(n^2)$	SRS+PKI
BRandPiper [13]	sync.	BFT-SMR	VSS	D	$O(fn^2)$	1	✓	$O(n^2)$	$O(n^2)$	SRS+PKI
SPURT [27]	partial sync.	leader QC	PVSS	D	$O(n^2)$	1	✓	$O(n^2)/O(n)$	$O(n)$	PKI
OptRand [14]	sync.	HotStuff-SMR	PVSS	D	$O(n^2)$	1	✓	$O(n)$	$O(n)$	PKI
GRandLine [28]†	async.	BA	PVSS	D	$O(n^2)$	1	✓	$O(n)$	$O(n)$	DKG
Rondo [15]	partial sync.	Rondo-BFT	BAVSS	D	$O(n^2 \log n)$	1	✓	$O(n)$	$O(n)$	CRS+PKI
RandHerd [60]	async.	BFT-CoSi	PVSS	Cond.	$O(nc^2 \log n)$	1	✓	$O(c^2 \log n)$	$O(1)$	DKG
HashRand [37]	async.	Gather+ϵ-AA	BAwVSS	MC	$O(n^2 \log n)$	1	✓	$O(n \log n)$	–	secure ch.
AsyRand	async.	RBC+ABA	PVSS	LV	$O(n^2)^{\ddagger}$	$f+1$	✓	$O(n)$	$O(n)$	PKI

Coord. identifies coordination algorithms. Crypto. identifies core cryptographic primitive. Unpred. gives the epoch distance beyond which outputs are guaranteed unpredictable. Comm. counts the transferred bits per beacon output, suppressing security parameters. Comp./Verif. count per-party group operations per output, except HashRand's computation entry, which counts hash evaluations. SPURT lists leader/other-party costs. For RandHerd, c denotes the group size. "–" means that no such cost is specified for the listed instantiation. HashRand clients collect $f+1$ matching authenticated outputs. Its optional PKI-based signature extension is not included. †: GRandLine requires synchrony for preprocessing, comprising DKG and a one-time commitment phase. Each subsequent beacon output requires one asynchronous communication round. Its BA coordination is used during DKG. ‡: Optimistic generation cost: unanimous honest input 1 to the whole-column ABA gives constant rounds. The general worst-case expected amortized communication bound is $O(n^2 2^n)$. D: deterministic termination under the stated network assumptions. Cond.: progress subject to an additional condition.

disagreement probability, respectively. HashRand [37] reports a hash-based asynchronous beacon using batched weak secret sharing, Gather to establish a common core of completed sharings, and approximate agreement over pairwise secure channels. It distinguishes deterministic, Monte Carlo (MC), and Las Vegas (LV) termination. MC allows disagreement after fixed rounds, whereas LV preserves agreement with a random completion time. AsyRand follows the LV style, providing agreement and almost-sure completion using local-coin ABA.

Randomness for asynchronous agreement. Rabin [43] supplies common randomness through predistributed secrets. Canetti and Rabin [46] generate it through asynchronous sharing. Cachin et al. [47] derive it from an established threshold key. Asynchronous DKG constructions by Kokoris-Kogias et al. [19], Das et al. [20], and Abraham et al. [17] can bootstrap through intermediate sharing and agreement. Their control randomness need not depend on the completed target DKG. More recently, Abraham et al. [22] construct a strongly adaptively secure ADKG protocol for $f < n/3$, assuming a plain PKI, random oracles, and secure erasures. Their recursive construction bootstraps weak coins from small-group consensus and combines aggregated PVSS transcripts with Weighted Gather to supply randomness for agreement at larger scales. Gao et al. [50] construct common randomness with PKI alone and describe a DKG-free asynchronous beacon obtained by repeatedly running their ABA-based leader election and extracting bits from the selected VRF output. Thus DKG avoidance is already available through other asynchronous constructions. Local-coin ABA offers another control mechanism. Bracha's protocol [45] and WaterBear [58] use independent local randomness. WaterBear's Quadratic-ABA reduces messages per round to quadratic, while its worst-case expected round count remains exponential.

Maintaining efficiency in beacons. Beacon protocols reduce overhead through aggregation, batching, grouping, or limiting the contributions opened per output. SPURT [27] uses homomorphic PVSS aggregation: a leader combines commitments and encrypted shares, parties check disjoint parts of the aggregation, and agreement operates on its digest to reduce broadcast traffic. Rondo [15] shares batches of secrets through batched asynchronous VSS with partial output, using each batch across multiple beacon rounds. HashRand [37] batches weak secret sharing and pipelines preparation to amortize sharing and broadcast costs across outputs. RandHerd [60] randomly partitions parties into smaller secret-sharing groups and combines their threshold signatures through a communication tree, reducing per-node overhead subject to sufficient honest participation in every group. Following synchronous DKG and a one-time commitment phase, GRandLine [28] produces each beacon in a single asynchronous communication round by reconstructing a threshold signature from verifiable signature shares and hashing it, without a new PVSS sharing phase for each output. Under synchronous coordination, HydRand [29] and GRandPiper [13] derive each output from one previously committed PVSS secret together with prior beacon state, avoiding reconstruction of multiple parties' contributions for every output. AsyRand also consumes one accepted PVSS object per output, while amortizing column dissemination, agreement, and cached verification across outputs in a fully asynchronous execution.

3. Building Blocks

3.1. Publicly Verifiable Secret Sharing

A publicly verifiable secret sharing (PVSS) scheme [24], [25] allows a dealer to commit to a hidden value that can later be reconstructed from t valid reconstruction shares. We write (C, π) for the sharing transcript, $Z = g^s$ for its bound value in a prime-order group $G = \langle g \rangle$, and D_i for party i's reconstruction share. Here λ is the security parameter

and $|G| = p$. The PVSS scheme exposes the following algorithms. Public parameters and PVSS public keys are implicit after setup unless explicitly listed.

1) $(pp, \{(sk_i, pk_i)\}_{i\in[n]}) \leftarrow \mathsf{PVSSSetup}(\lambda, t, n)$ creates public parameters and one PVSS key pair per party.
2) $(C, \pi) \leftarrow \mathsf{PVSSShare}(s)$ takes a scalar $s \in \mathbb{Z}_p$, producing public data C and proofs π that bind the group value $Z = g^s$.
3) $ok \leftarrow \mathsf{PVSSVerify}(C, \pi)$ publicly checks the dealer transcript in the sharing.
4) $D_i \leftarrow \mathsf{PVSSPreRecon}(C, sk_i)$ produces party i's publicly checkable reconstruction share without revealing sk_i.
5) $ok \leftarrow \mathsf{PVSSVerifyKey}(C, pk_i, D_i)$ publicly verifies that D_i belongs to party i and transcript C.
6) $Z \leftarrow \mathsf{PVSSRecon}(C, \{D_i\}_{i\in T})$ combines the valid shares indexed by $T \subseteq [n]$, where $|T| \geq t$, using any t of them to recover the unique bound group value.

Appendix B instantiates these interfaces. The PVSS scheme has the following properties:

- **Correctness.** Honest sharing and reconstruction shares pass verification, and any t distinct valid shares of an honest sharing of s reconstruct $Z = g^s$.
- **IND1-secrecy.** For an honest sharing of uniform $s \in \mathbb{Z}_p$, an adversary given the public sharing transcript and the PVSS private keys of fewer than t parties cannot distinguish $Z = g^s$ from an independent uniform element of G before reconstruction.
- **Public verifiability.** Anyone can verify sharing transcripts (C, π) and every reconstruction share D_i. Even for a faulty dealer, every accepted transcript allows honest parties to generate valid shares, and any t distinct valid shares reconstruct the same value, except with negligible probability.

3.2. Reliable Broadcast

Reliable broadcast (RBC) lets a designated dealer disseminate a payload v consistently in an asynchronous Byzantine network. We use Bracha RBC [44], whose ECHO/READY protocol provides validity checking and transferable delivery evidence. Following the large-message separation of Cachin et al. [51], digest-based control and direct payload dissemination cost $O(n^2 + n|v|)$ bits, suppressing the security parameter. Payload recovery adds its actual transfer cost (Appendix C). A faulty dealer's instance may remain undelivered. Appendix C gives the construction and message details. For a PVSS object $O = (i, k, C, \pi)$ as the payload, the pair (i, k) identifies the broadcast instance and is bound by $H(O)$. We model RBC with following interfaces:

$$\mathsf{RBC.Broadcast}(v), \quad \mathsf{RBC.Deliver}(v, \mathsf{RBCCert}_{\mathsf{id},H(v)}).$$

The delivery certificate is transferable and binds the payload digest to its broadcast instance. A party delivers v only after checking its application validity. For a PVSS object, it checks $\mathsf{PVSSVerify}(C, \pi) = 1$. RBC provides:

- ***Validity.*** If an honest dealer broadcasts a value that passes the enclosing protocol's validation, every honest party eventually delivers it.
- ***Agreement.*** If two honest parties deliver values in the same broadcast instance, then those values are identical.
- ***Totality.*** If one honest party delivers a value, every honest party eventually delivers that value.

3.3. Asynchronous Binary Agreement

Asynchronous binary agreement (ABA) lets parties propose binary inputs and agree on one bit despite Byzantine faults. We use WaterBear's Quadratic-ABA [58] with fresh, independent local coins for $n > 3f$. Its execution requires no beacon-derived or external common coins. For each unique instance id, an honest party submits at most one input $b_i \in \{0, 1\}$. We model ABA with following interfaces:

$$\mathsf{ABA}_{\mathsf{id}}.\mathsf{Propose}(b_i), \quad \mathsf{ABA}_{\mathsf{id}}.\mathtt{Decide}(d, \mathsf{ABACert}_{\mathsf{id},d}).$$

Here `Decide` returns the decided bit and its certificate $\mathsf{ABACert}_{\mathsf{id},d}$, bound to instance id and decision d. Appendix D specifies the core of Quadratic-ABA.

The ABA interface satisfies:

- ***Validity.*** If all honest parties propose b, every valid ABA decision certificate certifies bit b.
- ***Honest-input support.*** A valid ABA decision certificate $\mathsf{ABACert}_{\mathsf{id},b}$ implies that some honest party proposed b in that instance before the certificate was formed. In particular, an ABA decision certificate for 1 requires an honest input 1.
- ***Agreement.*** No two valid ABA decision certificates for the same instance certify different bits.
- ***Integrity.*** An honest party delivers at most one decision per ABA instance.
- ***Totality.*** If an honest party delivers a decision with a valid ABA decision certificate, every honest party eventually delivers the same bit with that certificate.
- ***Termination.*** If every honest party eventually proposes and continues the prescribed core execution, every honest party obtains an ABA decision certificate with probability 1.

4. System and Security Model

4.1. System Model and Assumptions

Network and adversary. The fixed party set is $[n] = \{1, \ldots, n\}$, with $n \geq 2$ and a integer fault bound $0 \leq f < n/3$. The public setup consists of the PVSS parameters pp, PVSS public keys $\{pk_i\}_{i\in[n]}$, the fault threshold f, and the specified ABA rules. Authenticated point-to-point channels provide sender authentication and integrity under a PKI. Honest-to-honest messages are eventually delivered, without a delay bound. RBC deliveries or ABA decisions may arrive out of order when multiple RBC or ABA instances are initialized in parallel. A PPT rushing adversary statically

corrupts at most f parties before setup and controls them for the entire execution. It may delay, reorder, and replay messages, observe honest messages before choosing faulty messages, and withhold faulty-party messages. Every continuously enabled local protocol action at an honest party is eventually executed, including actions of the Producer, Consumer, and active RBC/ABA instances. Honest local coins are fresh and independent across parties, instances, and invocations. At the protocol level, we assume the three PVSS properties of Section 3.1, EUF-CMA-secure signatures, and collision-resistant H. All hash inputs use fixed, unambiguous canonical encodings. With domain separation, H_{beacon}, H_{select}, H_{PVSS}, and H_{DLEQ} are modeled as independent random-oracle domains.

Objects and roles. Each party i holds a PVSS key pair (sk_i, pk_i) and runs both the Producer and replicated Consumer processes. Producer i's stream contains PVSS objects $O_{i,k} = (i, k, C, \pi)$, and column k contains the equally indexed objects of all Producers. The transcript (C, π) commits to a hidden, publicly verifiable value, and its RBC instance identifier binds (i, k). PVSS calls implicitly bind (i, k), so proofs cannot be reused at another stream position. Consumers buffer verified objects even if RBC delivers them out of order. Each party maintains one current consumption-column pointer κ, initialized to 1 before either process starts. Only the Consumer advances it, after all accepted objects in that column have been consumed. Different parties may have different local column indices during asynchronous execution. The production-window size is a finite integer $W \geq 2$. Producer i's next unreserved position p_i, initially 1, is enabled whenever $p_i \leq \kappa + W - 1$, independently of Consumer waits. Producers replenish this column-based window as their local Consumer advances κ and reserved positions are never re-sampled. Consuming one object does not individually enlarge its Producer's window.

Epochs and history. We use e for output epochs, k for columns, κ for the local current consumption column, and r for rounds within an ABA instance. For each epoch $e \geq 1$, R_e denotes the beacon value, and P_e denotes the corresponding history record. The record contains R_e and binds it to the preceding record, the consumed PVSS object, and the relevant column decisions. Honest Consumers install P_e and output R_e in increasing epoch order, although their local progress may differ. A single column k may yield multiple consecutive epochs, one per accepted object. The reconstruction threshold is $t = f + 1$. The quorum size for RBC delivery certificates and ABA decision certificates is $q = n - f$. Consumers maintain a queue *RCQ* of the most recent contributor identifiers, with capacity $f + 1$. The eligible Producer list *EPL* orders the Producers with accepted, unconsumed objects in the current column, excluding recent contributors recorded in *RCQ*.

4.2. Security Goals

Let $\mathcal{R} = \{0,1\}^\lambda$ and $\mathcal{H}_i = (P_0, \ldots, P_{e_i})$ be party i's installed genesis-and-beacon history. The relation $\preceq$ denotes prefix order. The public verifier $\mathsf{Verify}(P_e, \mathcal{W}_e; P_0)$ uses evidence $\mathcal{W}_e$, consisting of the preceding history and supporting ABA, RBC, and PVSS evidence, to check P_e by replaying the deterministic protocol transitions from genesis P_0. For $e \geq 1$, let $\mathsf{Complete}(e)$ mean that every honest party eventually installs the same valid P_e and outputs R_e. For $e \geq 0$, τ_e denotes the first honest installation of P_e, where P_0 is the genesis record. The adversarial view V_e is taken immediately after the first honest installation of P_e at τ_e.

Definition 1 (Security goals)**.** For every PPT adversary:

- **Safety.** Except with negligible probability, all honest histories are rooted at P_0, satisfy the protocol's publicly checkable transition rules, and are pairwise comparable under $\preceq$.
- **Public verifiability.** Every honestly installed record admits evidence $\mathcal{W}_e$ such that $\mathsf{Verify}(P_e, \mathcal{W}_e; P_0) = 1$, except with negligible probability. Moreover, no PPT adversary can produce a record and evidence accepted by the verifier while violating these transition rules, except with negligible probability.
- **Liveness.** With correct primitives, for every fixed $e \geq 1$, $\Pr[\mathsf{Complete}(e)] = 1$.
- **Unpredictability.** Fix $e \geq 0$ and $\ell \geq f + 1$ before execution. Any PPT adversary, using only V_e, produces a guess $\widehat{R} \in \mathcal{R}$ satisfying

  $$\Pr[\widehat{R} = R_{e+\ell}] \leq 2^{-\lambda} + \mathsf{negl}(\lambda).$$

  A match counts only if τ_e is reached and an honest party installs a valid record at epoch $e+\ell$ and outputs its beacon value.
- **Bias resistance.** Fix $e \geq 0$ and $\ell \geq f + 1$ before execution. Any PPT test D, given only V_e and either the eventual beacon output $R_{e+\ell}$ or an independent uniform value $R^\star \in \mathcal{R}$, satisfies

  $$\big|\Pr[D(V_e, R_{e+\ell}) = 1] - \Pr[D(V_e, R^\star) = 1]\big| \leq \mathsf{negl}(\lambda),$$

  If τ_e is not reached, the test result in both terms is defined as 0. No subsequent protocol transcripts are provided to D.

5. Overview

The mechanisms in Figure 1 jointly enforce two requirements: acceptance decisions of PVSS objects must precede the relevant revelation, and waiting for those decisions must not prevent future objects from being produced. We explain how production, agreement, selection, and verified reconstruction satisfy these requirements.

5.1. Parallel Production and Consumption

A party plays the role of both a Producer and a replicated Consumer. PVSS binds each object to a hidden value, while Bracha RBC [44] in Producers ensures that Consumers obtain a consistent transcript. These properties let Consumers agree on objects without reconstructing their hidden values.

Grouping equal stream positions into columns gives Consumers a common processing boundary despite out-of-order delivery. Producers prepare future objects within their production windows while Consumers process accepted ones. With production window size $W \geq 2$, honest Producers can prepare successor-column objects without waiting for current-column reconstruction.

5.2. Look-Ahead Agreement with Local Coins

Consumers first try to accept an entire column through one whole-column ABA. A decision of zero invokes parallel object ABAs. For an object not yet available locally, a Consumer may propose 0 only after at least $q = n - f$ other object ABAs in the same column have certified decision 1. This rule preserves at least q accepted contributions before allowing missing objects to receive input 0, so delivery delays cannot leave the column with too few accepted objects. Once all required ABA decisions are certified, Consumers wait until every accepted object has been RBC-delivered and its PVSS transcript verified. The ABA decisions determine which objects will be consumed. RBC ensures that honest Consumers obtain the same transcript for each accepted object. Fresh local coins allow these control instances to progress without beacon-derived randomness.

The look-ahead barrier requires complete successor-column decisions and accepted objects before any current-column reconstruction share is released. Consumers then use the previous beacon to select an unconsumed object deterministically, excluding recent contributors across column boundaries. The exclusion ensures that any sequence of $f+1$ consecutive outputs includes an honest contribution. These mechanisms, together with PVSS IND1-secrecy and beacon chaining, support the future-output security guarantees.

At startup, Consumers derive a common public initial value R_0 from the first column's agreed decisions and object hashes. Each beacon output is then constructed locally from an accepted object and verified reconstruction shares, without a separate agreement on the output.

6. The AsyRand Protocol

The Overview gives the design rationale. This section specifies the executable protocol. Process 1 starts the Producer and Consumer algorithms concurrently under the public setup and primitive interfaces already specified. The following subsections define each role's steps.

Process 1: The AsyRand process at each party i.

1 $\kappa \leftarrow 1$ (shared consumption column)
2 **asynchronously start** *Producer* i: Process 2
3 **asynchronously start** *Consumer* i: Process 3

6.1. The Producer Process

Producer i maintains a local sequence counter p_i, initialized to 1, for the next object it will reserve. The shared κ denotes its Consumer's current column. The Producer only reads κ. Its configured window $W \geq 2$ permits production while $p_i \leq \kappa + W - 1$. The PVSS proofs are bound to the Producer identifier and stream position (i, p_i). Advancing κ replenishes the window. Until then, the Producer eventually reserves each enabled position without waiting for agreement or reconstruction.

Process 2: The Producer process at each party i.

1 **Initialization.**
2 $p_i \leftarrow 1$
3 **Step 0: produce and broadcast.**
4 **upon** $p_i \leq \kappa + W - 1$
5 sample $s \xleftarrow{\$} \mathbb{Z}_p$; $(C, \pi) \leftarrow \mathsf{PVSSShare}(s)$
6 $O \leftarrow (i, p_i, C, \pi)$; $p_i \leftarrow p_i + 1$
7 $\mathsf{RBC.Broadcast}(O)$
8 **Step 1: deliver and store.**
9 **upon** $\mathsf{RBC.Deliver}(O, \mathsf{RBCCert}_{H(O)})$ *for* $O = (j, k', C, \pi)$
10 assert $\mathsf{PVSSVerify}(C, \pi) = 1$
11 $\mathit{ObjQ}[j][k'] \leftarrow (O, \mathsf{RBCCert}_{H(O)})$

Process 2 has two event handlers. When the production window has capacity, Line 5 samples a value and creates its PVSS transcript. Line 6 forms the object using the current p_i and increments the counter. Line 7 RBC-broadcasts the object. Upon RBC delivery, Line 10 verifies the PVSS transcript, and Line 11 stores $O = (j, k', C, \pi)$ with its delivery certificate in $\mathit{ObjQ}[j][k']$. A missing entry is $\perp$.

A faulty Producer may withhold an object, provide an invalid PVSS transcript, or send conflicting objects for the same position (i, k). Honest parties reject invalid transcripts, and RBC prevents them from delivering different objects at that position. Missing objects are handled by the Consumer's column-agreement procedure, which allows progress without requiring every Producer to respond. Once an object is accepted, Consumers can reconstruct its value without further cooperation from its Producer.

6.2. The Consumer Process

Process 3 decides columns and consumes their accepted objects. For column k, the procedure $\mathsf{DecideColumn}(k)$ determines which objects are accepted and constructs the column decision record A_k, containing each object's acceptance status. Each Consumer starts the procedure at most once per column. Repeated calls do not start another execution and callers still wait for A_k. The instance identifiers $\mathsf{col}(k)$ and $\mathsf{obj}(k, S)$ distinguish the whole-column ABA for column k and the object ABA for Producer S in that column. For column k, the set $\mathsf{ready}(k) = \{S : \mathit{ObjQ}[S][k] \neq \perp\}$ contains RBC-delivered, PVSS-verified objects. The flag $\mathsf{begun}(k)$ is initially false. It becomes true when the Consumer starts preparing to consume the first object in column k, and remains true thereafter. The flag $\mathsf{enough}(k)$ is set to

true once the Consumer has verified decision-1 certificates for at least q distinct objects in column k.

Startup. The shared $\kappa = 1$ enables production while the Consumer waits for A_1. Once this record A_1 is complete and verified, the Consumer derives the public seed R_0 and genesis P_0 (Lines 18–19), installs genesis, and initializes $e = 1$, $U = C_1$, and $\mathit{RCQ} = \emptyset$. At each epoch, P_{e-1} and R_{e-1} are read from the latest installed record, with R_0 used at startup.

Column agreement. The whole-column ABA receives input 1 when all n objects are ready. Input 0 requires at least $2f + 1$ ready objects and, for $k > 1$, $\mathsf{begun}(k - 1)$ (Lines 4–5). The indicator $d_S \in \{0, 1\}$ specifies whether object $O_{S,k}$ is accepted. The whole-column decision 1 sets all $d_S = 1$ without object ABAs. Decision 0 invokes them in parallel. Each object ABA receives input 1 for a ready object, while a Consumer may propose 0 for an unavailable object only when $\mathsf{enough}(k)$ is true. For either kind of ABA, a Consumer may accept a decision after verifying its certificate (Line 11), even if it has not yet submitted an input to that instance.

After obtaining all required ABA decision certificates, the Consumer defines $C_k = \{S \in [n] : d_S = 1\}$, the set of Producers whose objects in column k are accepted, and checks $|C_k| \geq q$ (Line 14). It then waits until every accepted object has been RBC-delivered and its PVSS transcript verified (Line 15). Finally, it constructs the column decision record A_k (Line 16). Thus, A_k fixes both the accepted set and the contents of its objects.

Step 0: select an object and release a share. The Consumer maintains $U \subseteq C_\kappa$, the set of Producers whose accepted objects in the current column remain unconsumed. It also maintains the recent-contributor queue *RCQ* of capacity $f + 1 < n$. The installed history determines both states. On the first output of a column, identified by $U = C_\kappa$, the Consumer sets $\mathsf{begun}(\kappa)$ and waits for complete $A_{\kappa+1}$ before releasing a share (Line 27). With $W \geq 2$, successor production proceeds independently of this wait.

The Consumer forms the eligible Producer list $\mathit{EPL} = \mathsf{sort}(U \backslash \mathsf{set}(\mathit{RCQ}))$, indexed from 0 (Line 28). Using epoch e and the previous value R_{e-1}, it selects a Producer identifier S from this list (Line 29). It retrieves the corresponding object $O_{S,\kappa}$ and checks its hash against the entry for S in A_κ (Line 30). The Consumer then generates its own reconstruction share D_i for this object and broadcasts it together with $(e, H(P_{e-1}), \kappa, S)$ (Lines 31–32). Thus selection excludes consumed objects and recent contributors across column boundaries.

Step 1: reconstruct and advance. Verified decisions, the selected object, and t valid shares from distinct indices $T \subseteq [n]$, $|T| = t$, enable reconstruction (Line 35). The Consumer sends its own share first if needed under Step 0's checks, computes R_e, and forms P_e (Line 38). It removes S from U, drops *RCQ*'s oldest entry if full, and always enqueues S, then outputs and advances the epoch (Line 39-41). If U is empty, it increments the shared κ and initializes $U = C_\kappa$ from the verified successor (Line 43). Record installation, these state updates and output form one atomic transition, completed before the next Step 0 starts.

Remark on Execution and recovery. Consumers retain records, accepted objects, certificates, and reconstruction shares, and provide them on request. A lagging Consumer verifies this evidence and recomputes missed records in epoch order, accepting a retrieved record only if it matches recomputation. Before releasing a share, it follows the same predecessor-installation and column-decision requirements as normal execution, and broadcasts any unsent local share before installing the record. Reconstruction-share broadcasts and record installation occur at most once per Consumer per epoch.

6.3. Optimization: Parallel ABA and Consumption

Process 3 waits for the successor ABA decisions before consuming the first object. A parallel variant can postpone this wait: before consuming column k's first accepted object, set $\mathsf{begun}(k)$ and start $\mathsf{DecideColumn}(k + 1)$ in the background. Consumers process the first f accepted objects while that procedure runs. Before releasing any honest reconstruction share for the $(f + 1)$-st object, they wait for complete, verified A_{k+1}. For the first f consumed objects of column k, the successor reference is always $\perp$. Thereafter it is $H(A_{k+1})$. Share release, including during recovery, requires verified A_k and, from the $(f+1)$-st object onward, verified A_{k+1}. Record construction and public verification derive positions from the verified history and apply these reference and decision rules. All other protocol rules remain unchanged.

The parallel variant relaxes the look-ahead barrier of Lemma 1 while preserving the *window-binding property*: for any $f + 1$ consecutive outputs, the relevant accepted objects and decisions are fixed before the first honest reconstruction share for those outputs is released. Indeed, $|C_k| \geq n-f \geq 2f+1$. After the first f objects, at least $f+1$ remain, so the last f objects are consumed only after A_{k+1} is fixed. Any $f+1$-output window spans at most two columns. Within one column, its inputs are already fixed by that column's decision record. Across a boundary, at most f objects belong to the earlier column, all in its final f positions. Consequently, both decision records are fixed before the window's first honest reconstruction share is released. *RCQ* still guarantees distinct contributors, including one never corrupted. Together, these properties support Theorems 4–5. The deterministic record rule preserves agreement and public verification, while continued production with $W \geq 2$ and ABA preserve almost-sure progress. The variant adds no ABA instances or asymptotic communication cost.

6.4. Optimization: Reproposable Agreement

Object agreement could alternatively use reproposable asynchronous binary agreement (RABA), instantiated with WaterBear's Quadratic-RABA [58]. In the object-agreement branch, verified RBC delivery of $O_{S,k}$ triggers an input

Process 3: The Consumer at party i.

procedure $\mathsf{DecideColumn}(k)$
  if *a previous task or complete* A_k *already exists* **then**
    return
  wait until $|\mathsf{ready}(k)| = n$ or $((k = 1 \vee \mathsf{begun}(k-1)) \wedge |\mathsf{ready}(k)| \geq 2f+1)$, or adopt an existing ABA decision certificate
  if $\mathsf{ABA}_{\mathsf{col}(k)}$ has no certified decision, $\mathsf{ABA}_{\mathsf{col}(k)}.\mathtt{Propose}([|\mathsf{ready}(k)| = n])$
  wait for verified $\mathsf{ABACert}_{\mathsf{col}(k),b}$
  if $b = 1$ **then**
    $d_S \leftarrow 1$ for every $S \in [n]$
  else
    foreach $S \in [n]$ in parallel **do**
      wait until $S \in \mathsf{ready}(k)$, or $\mathsf{enough}(k)$ is true, or $\mathsf{ABA}_{\mathsf{obj}(k,S)}$ has a certified decision
      if $\mathsf{ABA}_{\mathsf{obj}(k,S)}$ has no certified decision, $\mathsf{ABA}_{\mathsf{obj}(k,S)}.\mathtt{Propose}([S \in \mathsf{ready}(k)])$
    wait for verified $\mathsf{ABACert}_{\mathsf{obj}(k,S),d_S}$ for every $S \in [n]$
  $C_k \leftarrow \{S : d_S = 1\}$; assert $|C_k| \geq q$
  wait until $C_k \subseteq \mathsf{ready}(k)$
  form $A_k = (k, b, (d_S)_{S\in[n]}, [(S, H(O_{S,k}))]_{S\in C_k})$

Startup: derive genesis from the first Consumer decision.
call $\mathsf{DecideColumn}(1)$ and wait for complete, verified A_1
derive $R_0 = H(A_1, pp, \{pk_i\}_{i\in[n]}, n, f)$ and $P_0 = (\mathrm{GENESIS}, H(A_1), R_0)$
initialize $e = 1$, $U = C_1$, $\mathit{RCQ} = \emptyset$, and $R_{e-1} = R_0$
start the next beacon (Step 0)

Step 0: select an object and release a share.
upon *starting the next beacon using the current state*
  obtain verified A_κ; derive U from the history
  if $U = C_\kappa$ **then**
    $\mathsf{begun}(\kappa) \leftarrow \mathsf{true}$
    call $\mathsf{DecideColumn}(\kappa+1)$ and wait for complete, verified $A_{\kappa+1}$
  $\mathit{EPL} \leftarrow \mathsf{sort}(U \setminus \mathsf{set}(\mathit{RCQ}))$
  $S \leftarrow \mathit{EPL}[H_{\mathsf{select}}(e, R_{e-1}) \bmod |\mathit{EPL}|]$
  read $\mathit{ObjQ}[S][\kappa] = (O, \mathsf{RBCCert}_{H(O)})$; verify $H(O)$ against A_κ
  $D_i \leftarrow \mathsf{PVSSPreRecon}(C, sk_i)$ for $O = (S, \kappa, C, \pi)$
  broadcast $(\mathrm{RECON}, e, H(P_{e-1}), \kappa, S, D_i)_i$

Step 1: reconstruct and advance.
upon *having verified* $A_\kappa, A_{\kappa+1}, O$ *and* t *valid shares for the current* $(e, H(P_{e-1}), \kappa, S)$
  check t shares with distinct indices T using $\mathsf{PVSSVerifyKey}$; $Z \leftarrow \mathsf{PVSSRecon}(C, \{D_j\}_{j\in T})$
  send the local RECON if needed (Step 0)
  $R_e \leftarrow H_{\mathsf{beacon}}(e \parallel R_{e-1} \parallel Z)$
  $P_e \leftarrow (\mathrm{BEACON}, e, H(P_{e-1}), \kappa, S, H(O), H(A_\kappa), H(A_{\kappa+1}), R_e)$
  $U \leftarrow U \setminus \{S\}$
  if $|\mathit{RCQ}| = f+1$, dequeue its oldest member; enqueue S
  output R_e; $e \leftarrow e+1$
  if $U = \varnothing$ **then**
    $\kappa \leftarrow \kappa+1$; $U \leftarrow C_\kappa$
  start the next beacon (Step 0)

update to its object-RABA instance. A Consumer that previously proposed 0 and has neither locally decided nor adopted a decision certificate invokes $\mathtt{Repropose}(1)$ exactly once upon verified RBC delivery, regardless of its current RABA round. Consumers that have already proposed or reproposed 1 do not repeat the update. Reproposals cannot change a locally decided or certified value and do not cancel required background help. Consumers still propose an initial 0 only after obtaining decision-1 certificates for q other objects in the same column. RABA decisions can use the same q-signature certificates. In the RABA variant, honest-input support includes both initial proposals and reproposals of 1, each requiring verified RBC delivery of the corresponding object. For $n = 3f+1$, Quadratic-RABA's biased validity guarantees that if at least $f+1$ honest Consumers initially propose 1, any decided value is 1. When all honest Consumers initially propose 1, it additionally provides a one-step core fast path. Each core round uses $O(n^2)$ messages. The algorithms, security proofs, and complexity analysis in this paper are based on the Quadratic-ABA instantiation.

6.5. Complexity Analysis

Each beacon consumes one accepted PVSS object. A column disseminates at most n objects and yields $m_c = |C_c| \geq n - f = \Omega(n)$ outputs, charging $n/m_c = O(1)$ objects per beacon, including rejected objects. All bounds amortize over complete columns. Advance production under W is charged to the objects' own columns. Computation and verification costs count group operations only, excluding hashes, finite-field arithmetic, and ordinary message processing. Appendix B's ALBATROSS PVSS object [26] contains $2n$ group elements and $t+1$ scalars. Sharing and transcript verification each use $2n$ exponentiations. A reconstruction share contains three group elements and one scalar, costs three exponentiations to generate and four to verify. Verifying $t = f + 1$ shares and reconstructing costs $5t$ exponentiations. PVSS uses no pairings.

Communication. For an $O(n)$-size object, Appendix C's digest-control RBC costs $O(n|O|+n^2) = O(n^2)$ aggregate bits, including direct payload dissemination and digest ECHO/READY messages. Reconstruction broadcasts one constant-size share per party, adding $O(n^2)$ bits per output. Each column first uses one whole-column ABA. In the optimistic case, all honest parties propose 1: the ABA decides in one core round and executes one helping round, accepts all n objects, and costs $O(n)$ amortized bits per beacon. Decision 0 invokes n parallel object ABAs, giving $n+1$ instances in total. Each Quadratic-ABA round costs $O(n^2)$ bits, with a conservative $O(2^n)$ worst-case expected-round bound per instance [58]. Applying the geometric-tail maximum argument for parallel agreements [41] through a union bound gives $O(2^n \log n)$ expected core rounds until all fallback instances finish, excluding RBC readiness and proposal delays. Communication sums instance rounds rather than taking their maximum, giving $O(n2^n)$ and $O(n^2 2^n)$ expected amortized ABA bits per beacon on the one-instance and fallback paths, respectively. RBC READY and ABA decision signatures contribute $O(n^2)$ amortized bits per beacon. Total generation communication is therefore $O(n^2)$ in the optimistic case, with an $O(n^2 2^n)$ upper bound in worst-case expectation.

Computation and verification. Checking n PVSS transcripts costs $O(n^2)$ exponentiations per party per column, or $O(n)$ per beacon. Sharing, reconstruction, and checking at most n distinct shares preserve this bound. RBC/ABA certificates require $O(1)$ signature generations and $O(n)$ verifications per party per beacon, preserving the bound for signatures using $O(1)$ group operations.

7. Security Proof

We prove the goals of Definition 1 using the three PVSS properties of Section 3.1, together with correct RBC/ABA interfaces, authentic signatures, collision-resistant hashes, and the independent random-oracle domains. Cryptographic failures in the analyzed instances add a negligible term. Almost-sure progress is stated for correct interfaces and $W \geq 2$.

7.1. Safety

An ABA decision certificate contains at least $q - f > 0$ honest signers. Since honest parties sign only local core decisions, the core's honest-input support extends to certificates. Two conflicting certificates would share an honest signer, since $2q - n > f$, contradicting the rule that an honest party signs at most one decision per instance.

Lemma 1. *Before any honest reconstruction share from column k, A_k, A_{k+1} are fixed. Valid histories are comparable, and every $f+1$ consecutive outputs have distinct contributors, including one identity never corrupted.*

Proof. ABA agreement fixes the whole-column bit and, when that bit is 0, every object bit through parallel object ABAs. RBC agreement fixes each accepted transcript, and PVSS public verifiability fixes its bound value. Hashes of decision records exclude evidence encodings, so independently assembled RBC and ABA certificates yield the same description. Before releasing its first reconstruction share for column k, a Consumer waits for and verifies A_{k+1}. Consumers catching up on missed outputs follow the same rule. Every later output retains that decision record. Starting from the unique genesis derived from A_1, induct on the history. The agreed column records and common prefix fix the eligible Producer list *EPL*. The epoch e and preceding beacon value R_{e-1} then determine the selected Producer S and object $O_{S,\kappa}$. RBC agreement and unique PVSS reconstruction then fix R_e and P_e, independently of the valid reconstruction subset. Deterministic state updates fix the next state. Thus every valid prefix has at most one valid successor, so valid histories are comparable. Across column boundaries, *RCQ* excludes each contributor for the next $f+1$ outputs, ensuring distinct contributors in every $f+1$-output window. At most f static corruptions therefore leave at least one never-corrupted contributor in each window. □

Theorem 1 (Safety). AsyRand *satisfies safety in Definition 1.*

Proof. Induct on honest installations from P_0. Given a valid installed prefix, a Consumer verifies the required decisions, selected object, and context-bound reconstruction shares, then computes P_e and updates its state atomically according to the prescribed transition. Recovery performs the same checks and installs a retrieved record only if it matches recomputation. Every honest history is therefore genesis-rooted and satisfies the publicly checkable transition rules.

For any honest parties i, j, the preceding argument makes $\mathcal{H}_i, \mathcal{H}_j$ valid. Lemma 1 then gives $\mathcal{H}_i \preceq \mathcal{H}_j$ or $\mathcal{H}_j \preceq \mathcal{H}_i$. Thus honest Consumers install the same record and output the same beacon value at every epoch they both reach, even when their history lengths differ. In particular,

$$\Pr[\mathcal{H}_i \not\preceq \mathcal{H}_j \wedge \mathcal{H}_j \not\preceq \mathcal{H}_i] \leq \mathsf{negl}(\lambda).$$

Both record validity and history comparability hold except with negligible probability of primitive failure, establishing safety. □

7.2. Public Verifiability

Theorem 2 (**Public verifiability**). AsyRand *satisfies public verifiability in Definition 1.*

Proof. **Completeness.** Every honestly installed record has supporting RBC delivery and ABA decision certificates, accepted PVSS objects, and t distinct verified reconstruction shares. These data and the retained history supply $\mathcal{W}_e$. The certificates pass verification, while PVSS correctness and public verifiability ensure that the shares reconstruct the same bound value used by the Consumer. Deterministic replay from P_0 gives $\mathsf{Verify}(P_e, \mathcal{W}_e; P_0) = 1$, except with negligible probability. **Soundness.** Certificate authenticity, hash collision resistance, and RBC/ABA agreement bind the accepted objects and decisions to their instances. PVSS public verifiability ensures unique reconstruction from t valid shares with distinct indices. After checking this evidence, deterministic replay from P_0 rejects any record inconsistent with the prescribed transitions. Invalid acceptance therefore occurs only with negligible probability of primitive failure. □

7.3. Liveness

Lemma 2. *Once all honest Consumers have installed genesis for $k = 1$, or completed column $k-1$ for $k > 1$, and begin preparing column k, the decision record A_{k+1} is completed almost surely and includes at least $q = n - f$ objects.*

Proof. When each honest Consumer begins preparation of column k's first output, its shared consumption column satisfies $\kappa = k$ and cannot decrease thereafter, so $\kappa \geq k$. The Producer's enabled upper bound therefore satisfies $\kappa+W-1 \geq k+W-1 \geq k+1$, since $W \geq 2$. If position $k+1$ has not already been reserved, sequential production eventually reaches it without waiting for consumption of column k. Honest Producers can thus supply the successor column independently of the Consumer's wait. Honest Producers eventually supply at least $n - f \geq 2f + 1$ honest objects of column $k + 1$. If a whole-column certificate already exists, totality supplies it. Otherwise every honest observer eventually meets the readiness gate and proposes, so ABA termination supplies a certificate almost surely. Decision 1 has an honest all-ready input. RBC totality supplies all n objects.

For decision 0, before the first honest object input 0, there must already be q ABA decision certificates for 1. If that threshold is never reached, no honest input 0 occurs. Every honest Producer's object eventually reaches all honest observers. In each such instance, observers either adopt an existing certificate for 1, or eventually all propose 1. Validity and termination give at least q such decisions, a contradiction. Once the threshold is reached, those certificates reach every honest observer. Every remaining instance either supplies an existing certificate or receives eventual honest participation and terminates almost surely. An ABA decision certificate for 1 has honest ready-input support, so RBC totality supplies every accepted transcript. Waiting for all decisions fixes an accepted set of size at least q. □

Theorem 3 (**Beacon liveness**). *Under the model of Section 4, correct primitives, and $W \geq 2$, every fixed $e \geq 1$ satisfies* $\Pr[\mathsf{Complete}(e)] = 1$.

Proof. Initially, $\kappa = 1$ enables column-1 production before genesis installation. Honest objects eventually satisfy the readiness gate, and the column-agreement argument of Lemma 2 gives A_1 almost surely, with at least q accepted objects. All honest Consumers then derive the same P_0 and initial state.

Let $m = |C_k| \geq q > f + 1$, and suppose j objects have been consumed. For $j < f+1$, at most $f+1-j$ Producers in *RCQ* belong to U, whereas $|U| = m-j > f+1-j$. For $j \geq f + 1$, every Producer in *RCQ* has already contributed in this column and is absent from U. Hence *EPL* $\neq \varnothing$ whenever $U \neq \varnothing$.

Induct on the epoch after all honest Consumers have installed its predecessor. Consumers eventually begin preparation, and Lemma 2 ensures that any successor-column wait ends almost surely. RBC/ABA totality and retained evidence eventually supply the accepted data and certificates to every honest Consumer. By PVSS correctness and public verifiability, each broadcasts a valid share for the same selected object before installation, including during recovery. Since $n-f \geq t$, eventual delivery or retrieval supplies every Consumer with a valid threshold subset. Each reconstructs the same value, installs the record, and outputs its beacon value. Column exhaustion advances κ, initializes the next accepted set, and replenishes the production window. Each fixed epoch requires only finitely many almost-surely terminating instances, so induction gives $\Pr[\mathsf{Complete}(e)] = 1$. □

7.4. Bias Resistance

Lemma 3 (Window binding). *For any target epoch $E \geq f + 1$, the last $f + 1$ outputs ending at E span at most two columns. Their accepted PVSS transcripts and ABA decisions are fixed before any honest reconstruction share from the earlier column is released.*

Proof. If an $f+1$-output window spanned three columns, it would contain all outputs of the middle column and at least one from each adjacent column, totaling at least $q+2 > f+1$ outputs—a contradiction. Thus, at most $f+1$ outputs span at most two columns. Let k denote the window's first column. Before the first honest reconstruction share from column k is released, the waiting rule requires complete decision records A_k and A_{k+1}. These records fix the accepted objects and ABA decisions in both columns, covering every column that the window may span. □

Theorem 4 (**Bias resistance**). *Fix $e \geq 0$, $\ell \geq f + 1$, and set $E = e + \ell$. Under the model of Section 4 and the PVSS properties, any PPT test D specified in Definition 1 satisfies*

$$\left|\Pr[D(V_e, R_E) = 1] - \Pr[D(V_e, R^\star) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Proof. If τ_e (the first honest installation time of P_e) is not reached, both test results are 0. Otherwise, no honest share for an epoch greater than e has been released at τ_e: each release requires prior genesis installation and installation/output of all preceding records, including during recovery. By Lemma 1, let u be the first epoch in $[E-f, E]$ with an honest contributor, and let Z_u be its object's bound value. Since $u \geq E-f > e$, its object's honest share release has not begun at τ_e. Lemma 3 fixes the window's accepted objects and decisions before its first honest reconstruction share.

With the PVSS private keys of at most $f = t-1$ parties compromised, IND1-secrecy hides Z_u before its object's first honest reconstruction share. An IND1 distinguisher checks whether its challenge value appears in the adversary's beacon queries. A uniform challenge matches with probability at most $Q_H/|G|$. If the real bound value matched with non-negligible probability, this would contradict IND1-secrecy. Here Q_H bounds the adversary's beacon queries. Summing the negligible query probabilities over polynomially many honest objects still gives a negligible bound. Thus, except with negligible probability, the input $u \parallel R_{u-1} \parallel Z_u$ has not been queried before its object's first honest reconstruction share. Its first query therefore returns a fresh uniform λ-bit value.

The fixed objects and deterministic selection prevent the adversary from changing contributions or choosing another branch after observing answers. For each subsequent epoch through E, queries recorded before its fresh predecessor value was sampled guess that value with probability at most $Q_H 2^{-\lambda}$. Otherwise, the next beacon query receives a fresh uniform answer. The epoch field prevents input reuse, and the independent selection oracle does not alter this bound. A union bound over the chain steps and queries makes prior hits and chain-value collisions negligible. Together with the IND1 advantage and primitive failures, this shows that R_E is computationally indistinguishable from an independent uniform value given V_e. □

7.5. Unpredictability

Theorem 5 (Unpredictability). *Under the same assumptions and notation as Theorem 4, the unpredictability bound of Definition 1 holds:*

$$\Pr[\widehat{R} = R_E] \leq 2^{-\lambda} + \mathsf{negl}(\lambda).$$

Proof. Define $D(V_e, x)$ to compute the adversary's guess $\widehat{R}$ using only V_e and return 1 iff $x = \widehat{R}$. It returns 0 if τ_e is not reached. Since $R^\star$ is independent and uniform, $\Pr[D(V_e, R^\star) = 1] \leq 2^{-\lambda}$. Theorem 4 therefore gives $\Pr[D(V_e, R_E) = 1] \leq 2^{-\lambda} + \mathsf{negl}(\lambda)$. The prediction success event additionally requires an honest installation of a valid target record and output of its beacon value, so its probability cannot exceed this bound. □

8. Implementation and Evaluation

We implement AsyRand in Go. For PVSS arithmetic, we use Kyber[1] for scalar and group operations and polynomial-sharing utilities. We additionally use `edwards25519`[2] for optimized arithmetic in public proof verification and reconstruction. Go's standard library provides Ed25519 signatures, SHA-256 hashing, and JSON serialization. Our authenticated TCP transport uses mutual TLS 1.3 through the standard-library `crypto/tls` package. We also implement SPURT [27] and GRandLine [28] using the same transport and worker pool. Their pairing operations use BLS12-381 from `bls12-381` v0.1.0.[3]

The experiments run on Amazon EC2 `t3a.medium` instances, each providing 2 vCPUs and 4 GiB of memory. We evaluate networks of $n \in \{16, 31, 61, 91, 121\}$ parties, with $f = (n-1)/3$. Each party runs on a dedicated EC2 instance. We distribute parties cyclically across eight regions: Canada (Central), Europe (Ireland), US West (N. California), US East (N. Virginia), US West (Oregon), US East (Ohio), Asia Pacific (Singapore), and Asia Pacific (Tokyo). Let $f_b \leq f$ denote the actual number of faulty parties. The main comparison uses $f_b = 0$, while the additional experiments use $f_b \in \{0, 1, f\}$ in Appendix E.

We configure a maximum production window of $W = 8$. The Producer dynamically adjusts its active window between 2 and W based on the reserve of fully supplied future columns and local processing and communication load. Cryptographic workers prioritize reconstruction over advance production. We cache verification results for PVSS transcripts and reconstruction shares. The Consumer implements the parallel variant in Section 6.3: successor-column agreement overlaps with reconstruction of the first f accepted objects in the current column. The experiments use Quadratic-ABA rather than RABA. Although RABA supports reproposals and a one-step optimistic fast path, its impact on latency and communication cost in our framework remains unevaluated.

Beacon throughput. AsyRand has the highest throughput for all five values of n in Figure 2(a). At $n = 16$, it produces 35.068 outputs/s, compared with 29.771 for GRandLine and 3.119 for SPURT. At $n = 61$, its throughput is 19.217 outputs/s, 2.11× GRandLine's 9.105 outputs/s. At $n = 121$, throughput falls to 7.654, 4.602, and 0.546 outputs/s for AsyRand, GRandLine, and SPURT, respectively. AsyRand's throughput is 1.66× that of GRandLine and 14.03× that of SPURT.

The throughput results reflect differences in online computation and coordination. AsyRand separates PVSS preparation from beacon generation and consumes one accepted PVSS object per output. When preparation keeps pace with consumption, throughput is governed primarily by the Consumer path, with column agreement amortized across multiple outputs. SPURT performs aggregation and agreement for

1. https://github.com/dedis/kyber, Version v3.1.0
2. https://github.com/FiloSottile/edwards25519/tree/v1.1.0
3. https://github.com/kilic/bls12-381

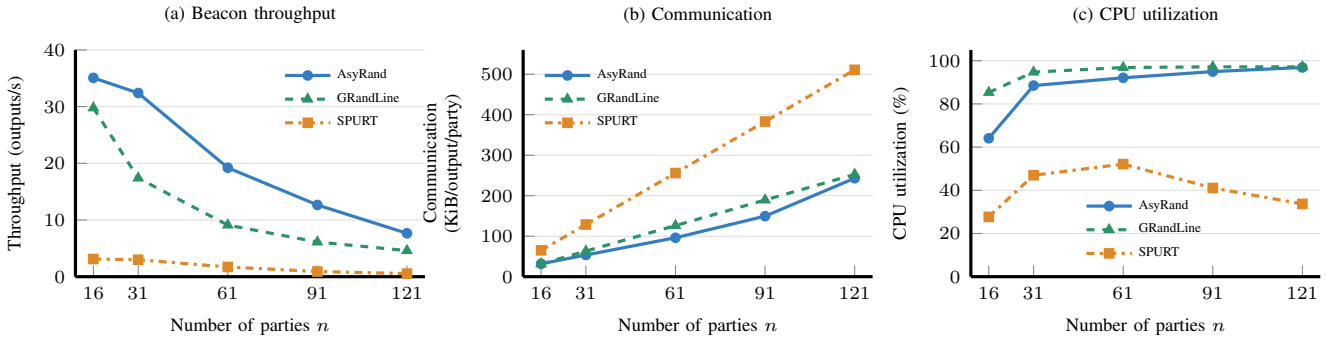

Figure 2. Beacon performance on AWS. (a) Throughput in outputs/s. (b) Sent communication in KiB/output per party. (c) CPU utilization per party, normalized to two vCPUs. Results are averaged across parties and three runs.

every beacon, introducing additional coordination between consecutive outputs. Following its PVSS-based DKG setup, GRandLine requires pairings for online share generation and verification, whereas AsyRand uses pairing-free reconstruction.

Communication cost. Figure 2(b) reports the data sent per output by each party. At $n = 61$, AsyRand sends 96.162 KiB/output, compared with 126.234 KiB for GRandLine and 256.006 KiB for SPURT, reductions of 23.8% and 62.4%. Across the five sizes, AsyRand uses 51.2–62.4% less communication than SPURT. Its communication is close to GRandLine's at $n = 16$ (31.739 versus 31.538 KiB/output) and lower at the remaining sizes. From $n = 61$ to $n = 121$, AsyRand's communication per output grows by a factor of 2.53, compared with 2.00 for GRandLine. This steeper increase accounts for the smaller communication saving at the largest size. At $n = 121$, AsyRand retains a $1.66\times$ throughput advantage with only 3.7% less data sent per output.

The communication results reflect the number of PVSS objects prepared per output, the amortization of agreement, and message size. SPURT aggregates at least $f + 1$ PVSS contributions for each beacon output. In AsyRand, a column prepares at most n objects and produces at least $n - f$ outputs, yielding an amortized $O(1)$ PVSS objects per output. Dissemination and agreement costs are shared across the column's outputs. Table 4 reports the ABA communication breakdown. GRandLine requires no per-output agreement after initialization, but its online share messages are larger in our implementation. At $n = 61$, an AsyRand reconstruction message averages approximately 933 bytes, compared with 2,154 bytes for a GRandLine share message. This difference offsets AsyRand's additional dissemination and agreement traffic. At $n = 16$, these costs nearly balance, resulting in similar communication per output.

CPU utilization and cost per output. Figure 2(c) reports CPU utilization normalized to each party's two-vCPU allocation. We also measure CPU time per output from cumulative CPU usage, including preparation, agreement, reconstruction, and runtime overhead. AsyRand's CPU utilization rises from 64.1% at $n = 16$ to 96.9% at $n = 121$, while its CPU time per output increases from 36.6 ms to 253.3 ms. Thus, parties become busier as each output requires more computation, even though throughput decreases. At $n = 121$, AsyRand and GRandLine both approach full CPU utilization, but their CPU times per output are 253.3 ms and 422.2 ms, respectively. AsyRand therefore uses 40.0% less CPU time per output, consistent with its $1.66\times$ throughput advantage under similar CPU utilization. SPURT uses 33.7% of the available CPU capacity at $n = 121$ but requires 1,235.0 ms of CPU time per output. Its lower utilization therefore does not imply lower computational cost per output. This combination is consistent with computation interspersed with waiting for communication and agreement.

9. Conclusion

AsyRand demonstrates how PVSS and local-coin ABA can support continuous asynchronous beacon generation without shared-key DKG or an external common coin. Look-ahead agreement fixes future accepted objects before current contributions are revealed, while recent-contributor exclusion ensures regular honest contributions. Under the stated assumptions, the protocol provides safety, public verifiability, and almost-sure completion, together with unpredictability and bias resistance for fixed targets at least $f+1$ epochs ahead. Our AWS evaluation shows the practical benefits of overlapping PVSS preparation with consumption and sharing agreement across multiple outputs. Without injected faults, AsyRand outperforms GRandLine and SPURT at all tested sizes. At 121 parties, it achieves $1.66\times$ GRandLine's throughput and $14.03\times$ SPURT's throughput.

Future work will explore the following directions: 1) Extend the framework's security proof from static to adaptive corruptions. 2) Support membership changes through handoffs at column boundaries, preserving the look-ahead barrier, the recoverability of retiring objects, and honest-contributor coverage across configurations. 3) Investigate using AsyRand's future outputs to supply common coins for randomized agreement in ADKG protocols.

References

[1] M. O. Rabin, "Transaction protection by beacons," *Journal of Computer and System Sciences*, vol. 27, no. 2, pp. 256–267, 1983.

[2] K. Choi, A. Manoj, and J. Bonneau, "SoK: Distributed randomness beacons," in *IEEE S&P*, 2023, pp. 75–92.

[3] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros Praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT*, 2018, pp. 66–98.

[4] T. Hanke, M. Movahedi, and D. Williams, "DFINITY technology overview series, consensus system," *arXiv:1805.04548*, 2018.

[5] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, "Single secret leader election," in *AFT*, 2020, pp. 12–24.

[6] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *SOSP*, 2017, pp. 51–68.

[7] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO*, 2017, pp. 357–388.

[8] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani, "Probabilistic smart contracts: Secure randomness on the blockchain," in *IEEE ICBC*, 2019, pp. 403–412.

[9] B. David, B. Magri, U. Matt, J. B. Nielsen, and D. Tschudi, "GearBox: Optimal-size shard committees by leveraging the safety-liveness dichotomy," in *CCS*, 2023, pp. 683–696.

[10] L. Luu, Y. Velner, P. Saxena, and A. Hobor, "Secure sharding on blockchains," in *CCS*, 2016, pp. 17–30.

[11] J. Van den Hooff, D. Lazar, M. Zaharia, and N. Zeldovich, "Vuvuzela: Scalable private messaging resistant to traffic analysis," in *SOSP*, 2015, pp. 137–152.

[12] D. Escudero, I. Tjuawinata, and C. Xing, "On information-theoretic secure multiparty computation with local repairability," in *PKC*, 2024, pp. 205–239.

[13] A. Bhat, N. Shrestha, Z. Luo, A. Kate, and K. Nayak, "RandPiper: Reconfiguration-friendly random beacons with quadratic communication," in *CCS*, 2021, pp. 3502–3524.

[14] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, "OptRand: Optimistically responsive reconfigurable distributed randomness," in *NDSS*, 2023.

[15] X. Meng, X. Sui, Z. Yang, K. Rong, W. Xu, S. Chen, Y. Yan, and S. Duan, "Rondo: Scalable and reconfiguration-friendly randomness beacon," in *NDSS*, 2025.

[16] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[17] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern, "Bingo: Adaptively secure packed asynchronous verifiable secret sharing and asynchronous distributed key generation," in *CRYPTO*, 2023, pp. 315–349.

[18] T. Yurek, L. Luo, J. Fairoze, A. Kate, and A. Miller, "hbACSS: How to robustly share many secrets," in *NDSS*, 2022.

[19] E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures," in *CCS*, 2020, pp. 1751–1767.

[20] S. Das, T. Yurek, Z. Xiang, A. Miller, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous distributed key generation," in *IEEE S&P*, 2022, pp. 2518–2534.

[21] S. Das, Z. Xiang, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling," in *USENIX Security*, 2023, pp. 5359–5376.

[22] I. Abraham, R. Bacho, J. Loss, and G. Stern, "Nearly quadratic asynchronous distributed key generation from recursive consensus," in *PODC*, 2026, pp. 247–258.

[23] H. Feng and Q. Tang, "Asymptotically optimal adaptive asynchronous common coin and DKG with silent setup," in *CRYPTO*, 2025, pp. 642–675.

[24] B. Schoenmakers, "A simple publicly verifiable secret sharing scheme and its application to electronic voting," in *CRYPTO*, 1999, pp. 148–164.

[25] I. Cascudo and B. David, "SCRAPE: Scalable randomness attested by public entities," in *ACNS*, 2017, pp. 537–556.

[26] I. Cascudo and B. David, "ALBATROSS: Publicly attestable batched randomness based on secret sharing," in *ASIACRYPT*, 2020, pp. 311–341.

[27] S. Das, V. Krishnan, I. M. Isaac, and L. Ren, "Spurt: Scalable distributed randomness beacon with transparent setup," in *IEEE S&P*, 2022, pp. 2502–2517.

[28] R. Bacho, C. Lenzen, J. Loss, S. Ochsenreither, and A. Papachristoudis, "GRandLine: Adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity," in *CCS*, 2024, pp. 941–955.

[29] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "HydRand: Efficient continuous distributed randomness," in *IEEE S&P*, 2020, pp. 73–89.

[30] J. Bonneau, J. Clark, and S. Goldfeder, "On Bitcoin as a public randomness source," *Cryptology ePrint Archive*, Paper 2015/1015, 2015.

[31] I. Bentov, A. Gabizon, and A. Zuckerman, "Bitcoin beacon," *Cryptology ePrint Archive*, Paper 2016/417, 2016.

[32] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *CRYPTO*, 2018, pp. 757–788.

[33] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, "RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness," in *NDSS*, 2021.

[34] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *ASIACRYPT*, 2001, pp. 514–532.

[35] C. Cachin, K. Kursawe, A. Lysyanskaya, and R. Strobl, "Asynchronous verifiable secret sharing and proactive cryptosystems," in *CCS*, 2002, pp. 88–97.

[36] V. Shoup and N. P. Smart, "Lightweight asynchronous verifiable secret sharing with optimal resilience," *Journal of Cryptology*, vol. 37, no. 3, Art. no. 27, 2024.

[37] A. Bandarupalli, A. Bhat, S. Bagchi, A. Kate, and M. K. Reiter, "Random Beacons in Monte Carlo: Efficient Asynchronous Random Beacon without Threshold Cryptography," *CCS*, 2024, 2621–2635.

[38] L. Freitas, P. Kuznetsov, and A. Tonkikh, "Distributed randomness from approximate agreement," in *DISC*, 2022, pp. 24:1–24:21.

[39] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Reaching consensus for asynchronous distributed key generation," in *PODC*, 2021, pp. 363–373.

[40] M. Ben-Or, "Another advantage of free choice: Completely asynchronous agreement protocols," in *PODC*, 1983, pp. 27–30.

[41] M. Ben-Or and R. El-Yaniv, "Resilient-optimal interactive consistency in constant time," *Distributed Computing*, vol. 16, no. 4, pp. 249–262, 2003.

[42] M. Ben-Or, B. Kelmer, and T. Rabin, "Asynchronous secure computations with optimal resilience (extended abstract)," in *PODC*, 1994, pp. 183–192.

[43] M. O. Rabin, "Randomized Byzantine generals," in *FOCS*, 1983, pp. 403–409.

[44] G. Bracha, "An asynchronous $(n-1)/3$-resilient consensus protocol," in *PODC*, 1984, pp. 154–162.

[45] G. Bracha, "Asynchronous Byzantine agreement protocols," *Information and Computation*, vol. 75, no. 2, pp. 130–143, 1987.

[46] R. Canetti and T. Rabin, "Fast asynchronous Byzantine agreement with optimal resilience," in *STOC*, 1993, pp. 42–51.

[47] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[48] A. Mostéfaoui, H. Moumen, and M. Raynal, "Signature-free asynchronous binary Byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time," *Journal of the ACM*, vol. 62, no. 4, pp. 31:1–31:21, 2015.

[49] I. Abraham, T.-H. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, "Communication complexity of Byzantine agreement, revisited," in *PODC*, 2019, pp. 317–326.

[50] Y. Gao, Y. Lu, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Efficient asynchronous Byzantine agreement without private setups," in *IEEE ICDCS*, 2022, pp. 246–257.

[51] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *CRYPTO*, 2001, pp. 524–541.

[52] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically optimal validated asynchronous Byzantine agreement," in *PODC*, 2019, pp. 337–346.

[53] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-MVBA: Optimal multi-valued validated asynchronous Byzantine agreement, revisited," in *PODC*, 2020, pp. 129–138.

[54] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *CCS*, 2016, pp. 31–42.

[55] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang, "Dumbo: Faster asynchronous BFT protocols," in *CCS*, 2020, pp. 803–818.

[56] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *PODC*, 2021, pp. 165–175.

[57] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: A DAG-based mempool and efficient BFT consensus," in *EuroSys*, 2022, pp. 34–50.

[58] H. Zhang, S. Duan, B. Zhao, and L. Zhu, "WaterBear: Practical asynchronous BFT matching security guarantees of partially synchronous BFT," in *USENIX Security*, 2023, pp. 5341–5357.

[59] E. MacBrough, "Cobalt: BFT governance in open networks," *arXiv:1802.07240*, 2018.

[60] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *IEEE S&P*, 2017, pp. 444–460.

[61] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "CoSi: Keeping authorities "honest or bust" with decentralized witness cosigning," in *IEEE S&P*, 2016, pp. 526–545.

Appendix A. Notations

Table 2 summarizes the party, beacon, and ABA notation used by the design and processes. Primitive calls implicitly use the fixed public parameters and authentication keys.

Section 4.2 defines processing histories $\mathcal{H}_i$, prefix order $\preceq$, and the first honest installation times τ_e with adversarial views V_e. Section 7.3 proves completion of column decisions and beacon outputs. Section 7.4 proves window binding and bias resistance.

TABLE 2. MAIN NOTATIONS IN THIS PAPER.

Symbol	Meaning
$[n], n, f$	fixed party set, size, and Byzantine threshold
$t = f + 1$, $q = n - f$	PVSS reconstruction threshold and quorum size for RBC delivery and ABA decision certificates
e, R_e, P_e	beacon epoch, beacon output, and beacon record
$\mathcal{W}_e$	history and supporting evidence used to verify record P_e
r	round index within an ABA instance
EPL, *RCQ*	eligible Producer list and recent-contributor queue of capacity $f + 1$
S, κ	selected Producer identifier and current consumption column
p_i	Producer i's next unreserved stream position
$W \geq 2$	production-window size
k	column index, also the stream position in each Producer's sequence
$O = (i, k, C, \pi)$	Producer i's k-th PVSS object
ObjQ$[i][k]$	local entry holding an RBC-delivered, verified object and its RBC delivery certificate
s, Z, D_i	sampled scalar, reconstructed group value, and party i's verified share
$\mathsf{RBCCert}_{\mathsf{id},H(v)}$	RBC delivery certificate binding payload v to instance id
$\mathsf{ABACert}_{\mathsf{id},d}$	ABA decision certificate binding bit d to instance id
$\mathsf{col}(k)$	fixed instance identifier for the whole-column ABA of column k
$\mathsf{obj}(k, S)$	fixed instance identifier for Producer S's object ABA in column k
A_k, C_k	complete decision record and set of Producers with accepted objects in column k
$U \subseteq C_\kappa$	Producers with accepted, unconsumed objects in the current column
$\mathsf{ready}(k)$	set of Producers with locally ready objects in column k
$\mathsf{begun}(k)$	flag indicating that consumption of column k has begun
$\mathsf{enough}(k)$	flag set after verifying decision-1 certificates from at least q distinct object ABAs in column k

Appendix B. The ALBATROSS PVSS Construction

We use the single-secret specialization of the DDH-based PVSS in ALBATROSS [26], which uses a low-degree exponent interpolation proof. Figure 3 presents the algorithms using the interfaces of Section 3.1. A degree-below-t polynomial shares one scalar s, and any t valid shares reconstruct $Z = g^s$. For AsyRand, $t = f + 1$. The implicit context χ binds the public parameters, ordered PVSS public keys, and sharing-instance identifier in both proof challenges.

Verification checks canonical encodings, group membership, scalar ranges, nonidentity public keys, and distinct reconstruction-share indices. The response polynomial has a fixed-length coefficient encoding, so its degree bound is

Functionality Single-secret ALBATROSS PVSS

$\mathsf{PVSSSetup}(\lambda, t, n), \quad 1 \le t \le n$:

Choose a DDH-hard group $G = \langle g \rangle$ of prime order p.

$\alpha_0 = 0$; fix distinct $\alpha_1, \ldots, \alpha_n \in \mathbb{Z}_p^*$.

$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, i)$:

$x_i \xleftarrow{R} \mathbb{Z}_p^*, \quad \mathsf{sk}_i = x_i, \quad \mathsf{pk}_i = g^{x_i}$.

$(C, \pi) \leftarrow \mathsf{PVSSShare}(s)$:

$F(X) = s + \sum_{j=1}^{t-1} \gamma_j X^j, \quad \gamma_j \xleftarrow{R} \mathbb{Z}_p$.

$C = (C_i)_{i=1}^n, \quad C_i = \mathsf{pk}_i^{F(\alpha_i)}$.

$F'(X) \xleftarrow{R} \mathbb{Z}_p[X]_{<t}, \quad C'_i = \mathsf{pk}_i^{F'(\alpha_i)}$.

$c = H_{\mathsf{PVSS}}(\chi, C, C'), \quad \widetilde{F} = F' + cF$.

$\pi = (C', c, \widetilde{F})$; encode $\widetilde{F}$ by its t coefficients.

$b \leftarrow \mathsf{PVSSVerify}(C, \pi)$:

Check $\deg \widetilde{F} < t$ and $c = H_{\mathsf{PVSS}}(\chi, C, C') \in \mathbb{Z}_p$.

Accept iff $\mathsf{pk}_i^{\widetilde{F}(\alpha_i)} = C'_i C_i^c$ for every $i \in [n]$.

$D_i \leftarrow \mathsf{PVSSPreRecon}(C, \mathsf{sk}_i)$:

$d_i = C_i^{1/x_i} = g^{F(\alpha_i)}, \quad w_i \xleftarrow{R} \mathbb{Z}_p$.

$a_i = g^{w_i}, \quad b_i = d_i^{w_i}$.

$z_i = w_i - x_i \cdot H_{\mathsf{DLEQ}}(\chi, i, C, d_i, a_i, b_i)$.

$D_i = (d_i, a_i, b_i, z_i)$.

$b \leftarrow \mathsf{PVSSVerifyKey}(C, \mathsf{pk}_i, D_i)$:

$\epsilon_i = H_{\mathsf{DLEQ}}(\chi, i, C, d_i, a_i, b_i) \in \mathbb{Z}_p$.

Accept iff $a_i = g^{z_i} \mathsf{pk}_i^{\epsilon_i}$ and $b_i = d_i^{z_i} C_i^{\epsilon_i}$.

$Z \leftarrow \mathsf{PVSSRecon}(C, \{D_i\}_{i \in T}), \quad |T| = t$:

$$\lambda_{i,T} = \prod_{j \in T \setminus \{i\}} \frac{\alpha_0 - \alpha_j}{\alpha_i - \alpha_j}.$$

$$Z = \prod_{i \in T} d_i^{\lambda_{i,T}} = g^{F(0)} = g^s.$$

Figure 3. Single-secret ALBATROSS PVSS [26], with context-bound proof challenges and the decryption challenge recomputed by the verifier.

checked directly. Proof nonces and honest sharing polynomials are fresh and independent. Correctness, public verifiability, and standard IND1-secrecy follow from the single-secret specialization of ALBATROSS [26].

Appendix C. Bracha Reliable Broadcast

Process 4 instantiates Bracha's echo/ready construction [44] with $q = n - f$. Each instance has a fixed dealer and identifier. For PVSS, these are Producer S and stream position (S, k). All messages bind the instance identifier, type, and fixed public setup $(pp, \{pk_i\}_{i \in [n]}, n, f)$. A party echoes at most one digest and sends at most one READY. Counts use distinct authenticated senders.

For payload v, write its digest as $\delta = H(v)$. The payload check requires this digest and application validity. For $v = O_{S,k}$, it checks the Producer identifier and stream position and verifies $\mathsf{PVSSVerify}(C, \pi) = 1$. An honest party sends ECHO only after this check. READY messages support the transferable RBC delivery certificate of Section 3.2. Delivery additionally requires the payload, so a digest alone cannot enter *ObjQ*.

Process 4: Bracha RBC with digest control at party i.

1 **State per instance:** *echoed*, *ready*, *delivered* $\leftarrow$ false
2 **upon** *the dealer invoking* RBC.Broadcast(v)
3 send $(\text{RBC-SEND}, v)$ to all parties
4 **upon** *a valid payload v from the dealer, if not echoed*
5 $\delta \leftarrow H(v)$; *echoed* $\leftarrow$ true
6 send $(\text{RBC-ECHO}, \delta)$ to all parties
7 **upon** *q matching ECHOs or $f+1$ matching READYs for δ, if not ready*
8 *ready* $\leftarrow$ true
9 sign and send $(\text{RBC-READY}, \delta)$ to all parties
10 **upon** *q matching signed READYs for δ, if not delivered*
11 assemble $\mathsf{RBCCert}_\delta$ from those READY signatures
12 obtain v, verify $H(v) = \delta$ and application validity
13 *delivered* $\leftarrow$ true
14 emit $\mathsf{RBC.Deliver}(v, \mathsf{RBCCert}_\delta)$

Two echo quorums intersect in more than f parties. Thus no two honest parties can originate READY for different digests. The $f + 1$ relay threshold cannot introduce a new digest without an earlier honest READY. A delivery quorum contains at least $n - 2f \ge f + 1$ honest READY senders. Their messages eventually make every honest party relay the same digest and collect q READYs. An honest dealer supplies validity directly.

Payload recovery. Honest ECHO senders retain validated payloads and answer retrieval requests. Honest receivers also retain delivered payloads and their delivery certificates and provide them on request. These storage operations are implicit in Process 4. If an RBC delivery certificate exists, tracing the first honest READY reaches q ECHOs, including honest payload holders. A party missing the payload requests it from the membership and accepts only a hash-matching valid response. Honest-to-honest delivery therefore supplies it eventually. This digest adaptation follows the large-payload separation of [51]. Its recovery traffic is charged separately. The uncompressed Bracha construction sends payloads in control messages and costs $O(n^2|v|)$ bits. Digest control costs $O(n^2)$, and direct dealer dissemination costs $O(n|v|)$. Arbitrary recovery is not assumed to preserve that latter bound.

Appendix D. WaterBear Quadratic-ABA

This appendix presents WaterBear's single-proposal Quadratic-ABA protocol [58]. The protocol operates over authenticated asynchronous channels and uses independent local coins. Each honest party proposes one binary input and locally decides a binary value. Process 5 gives the protocol

Process 5: Quadratic-ABA core at party i.

upon *the first* $\texttt{Propose}(b_i)$
  $r \leftarrow 0;\ a \leftarrow b_i;\ r_{\mathsf{stop}} \leftarrow \infty$
  while *true* **do**
    $\mathcal{B}_r \leftarrow \varnothing$
    send $\mathrm{PRE}(r, a)$ once to all parties
    activate persistent round handlers: relay $\mathrm{PRE}(r, v)$ once after $f+1$ senders; add v to $\mathcal{B}_r$ after $2f+1$ senders; validate pending messages
    wait for $\mathcal{B}_r \neq \varnothing$; $v \leftarrow \min \mathcal{B}_r$
    send $\mathrm{VOTE}(r, v)$ to all parties
    collect the first Q_{A} accepted VOTEs as V
    send $\mathrm{MAIN}(r, \mathsf{same}(V))$ to all parties
    collect the first Q_{A} accepted MAINs as M
    send $\mathrm{FINAL}(r, \mathsf{same}(M))$ to all parties
    if $r = r_{\mathsf{stop}}$ **then**
      return
    collect the first Q_{A} accepted FINALs as M_{final}
    if $\mathsf{vals}(M_{\mathsf{final}}) = \{v\}$ *for a bit* v **then**
      $a \leftarrow v$; emit $\texttt{CoreDecide}(v)$ once
      $r_{\mathsf{stop}} \leftarrow r + 1$
    else
      if $v \in \mathsf{vals}(M_{\mathsf{final}}) \subseteq \{v, *\}$ *for a bit* v **then**
        $a \leftarrow v$
      else
        $a \xleftarrow{\$} \{0,1\}$ independently
    $r \leftarrow r + 1$

at party i, where $\texttt{CoreDecide}(d)$ denotes a local decision on bit d.

Each party maintains a binary estimate a, initialized to its input b_i. The protocol proceeds in rounds, each involving four message phases: PRE, VOTE, MAIN, and FINAL. The latter three phases use a quorum of $Q_{\mathsf{A}} = \left\lceil \frac{n+f+1}{2} \right\rceil$ distinct parties. Any two such quorums intersect in at least one honest party, and the honest parties alone can form a quorum. When $n = 3f+1$, $Q_{\mathsf{A}} = 2f+1$. PRE uses separate relay and acceptance thresholds, as specified in Process 5.

Each sender broadcasts PRE once per bit and later phases once per round. Each later phase counts one accepted value per sender. Unsupported candidates are buffered without suppressing later valid ones, and future rounds wait for entry. Table 3 uses $v \in \{0,1\}$, accepted PRE bits $\mathcal{B}_r$, and wildcard $*$, allowed only in MAIN and FINAL. The function $\mathsf{same}(X)$ returns a batch's unanimous bit or $*$, and $\mathsf{vals}(X)$ its value set.

TABLE 3. LOCAL ACCEPTANCE PREDICATES IN QUADRATIC-ABA.

Message	**Required evidence in round** r
$\mathrm{VOTE}(v)$	$v \in \mathcal{B}_r$
$\mathrm{MAIN}(v)$	$f+1$ accepted $\mathrm{VOTE}(v)$
$\mathrm{FINAL}(v)$	$f+1$ accepted $\mathrm{MAIN}(v)$
$\mathrm{MAIN}(*)$, $\mathrm{FINAL}(*)$	$\mathcal{B}_r = \{0,1\}$

Decision and progress. A batch of unanimous binary FINAL messages causes a local decision on that bit. A batch containing one bit and the wildcard preserves that bit as the next estimate. Otherwise, the party samples a fresh independent local bit. After deciding, a party executes one additional round through FINAL transmission before stopping the main loop. Relay and validation handlers for entered rounds remain active to help other parties complete. Each round uses $O(n^2)$ messages. Unanimous honest inputs yield a decision in one core round, followed by one helping round. For general inputs, the local-coin analysis gives a conservative $O(2^n)$ worst-case expected-round bound.

Honest-input support. Before any honest proposal of bit b, faulty PREs cannot trigger an honest relay of b. Only $1-b$ can pass validation, excluding wildcards and random introduction of b in later rounds. Thus, a local decision on b requires an earlier honest proposal of b, even when proposals are staggered.

In AsyRand, an honest party signs only after locally deciding that bit at most once. Certificates are assembled locally from broadcast signatures, retained, and provided on request. An ABA decision certificate contains $q = n - f$ signatures from distinct parties.

Appendix E.
Additional Experimental Results

We additionally evaluate AsyRand with $f_b = 1$ or $f_b = f = (n-1)/3$ faulty parties, using the AWS setup and fault-free baseline of Section 8. In these runs, faulty parties send no protocol messages throughout execution. Figure 4 compares all three cases. With one faulty party, throughput retains 41.0–63.9% of the fault-free baseline and communication per output per honest party increases to 2.14–2.75 times the baseline. With f faulty parties, the corresponding ranges are 15.9–71.8% and 1.85–2.41 times. At $n = 121$, throughput is 7.654, 4.890, and 5.496 outputs/s with zero, one, and f faulty parties, respectively. Their communication costs are 243.158, 669.777, and 450.004 KiB/output/party. Figure 4(c) shows lower CPU utilization with $f_b = f$, whereas utilization with $f_b = 1$ remains close to the fault-free case at larger network sizes. At $n = 121$, utilization is 84.0%, 96.7%, and 96.9% for $f_b = f$, 1, and 0, respectively. With $f_b = f = (n-1)/3$, quorum formation requires all honest parties, making progress more sensitive to delayed responses. The lower utilization is consistent with more time spent waiting for messages. With $f_b = 1$, quorums can form without waiting for every honest party, while object-level agreement adds processing work, consistent with the observed high utilization.

In these runs, withholding even one Producer's contributions forces column agreement to fall back to object-level ABAs. With $f_b = f$ faulty parties, quorum collection also requires all $n - f$ honest parties, making progress more sensitive to message delays. Fewer active parties reduce dissemination and reconstruction traffic, so communication need not increase monotonically with the number of faulty

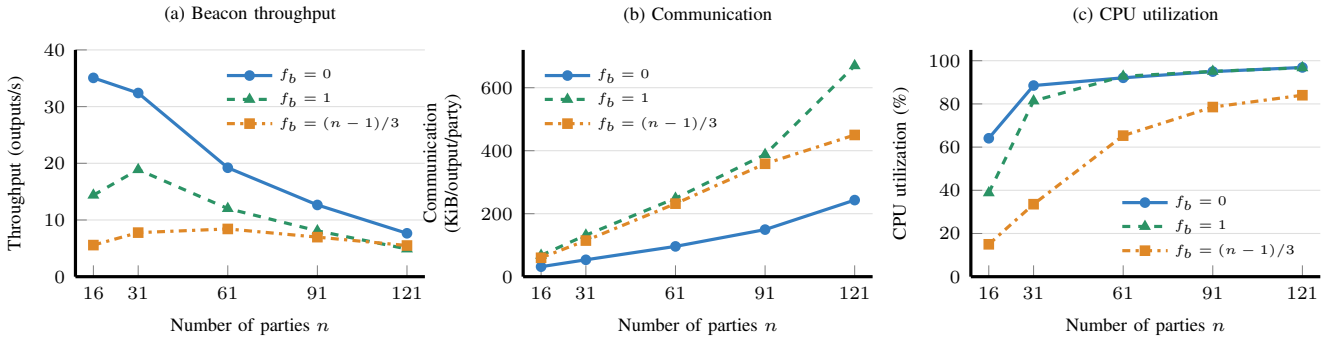

Figure 4. AsyRand with zero, one, or $f = (n-1)/3$ faulty parties. (a) Throughput in outputs/s. (b) Communication in KiB/output per honest party. (c) CPU utilization per honest party, normalized to two vCPUs. Results are averaged over honest parties and three runs. Here f_b counts faulty parties, while the configured fault bound remains $f = (n-1)/3$ in all cases.

parties. Table 4 reports total and ABA data sent per honest party per output. The final column is the ratio of mean ABA traffic to mean total sent traffic. Without faults, this ratio decreases from 7.19% at $n = 16$ to 0.97% at $n = 121$, consistent with amortizing column agreement across multiple outputs. With one faulty party, its proportion rises to 55.73–59.02%, compared with 66.12–70.65% when $(n-1)/3$ parties are faulty. At $n = 121$, one faulty party incurs more absolute ABA traffic than the maximum-fault case (395.309 versus 297.524 KiB/output/honest party), despite its smaller percentage (59.02% versus 66.12%). The percentage alone therefore does not measure absolute agreement cost. The $n = 121$ runs with faulty parties include a transport fix for large-frame write deadlines, as recorded in the experiment provenance.

TABLE 4. ABA COMMUNICATION FOR FIGURE 4. HERE f_b COUNTS FAULTY PARTIES. SENT VOLUMES ARE IN KIB/OUTPUT/HONEST PARTY.

n	f_b	**Total sent**	**ABA sent**	**ABA traffic (%)**
16	0	31.739	2.283	7.19%
	1	67.833	39.977	58.94%
	5	60.066	39.812	66.28%
31	0	53.606	2.382	4.44%
	1	131.892	75.388	57.16%
	10	114.696	79.285	69.13%
61	0	96.162	2.104	2.19%
	1	248.930	138.717	55.73%
	20	231.892	160.791	69.34%
91	0	149.465	1.925	1.29%
	1	387.799	219.614	56.63%
	30	358.550	253.327	70.65%
121	0	243.158	2.351	0.97%
	1	669.777	395.309	59.02%
	40	450.004	297.524	66.12%