AsyRand: Asynchronous Distributed Randomness Beacon with Reconfiguration

***Abstract*—Distributed randomness beacon protocols, which continuously generate publicly verifiable randomness values, are crucial for many applications. Recently, there have been many approaches, such as Hydrand (S&P'20), SPURT (S&P'22), OptRand (NDSS'23) and GRandLine (CCS'24), based on publicly verifiable secret sharing (PVSS) to implement beacon protocols. However, two key challenges remain unresolved: asynchrony and reconfiguration. In this paper, we propose the AsyRand beacon protocol to address these challenges. We incorporate a producer-consumer model to decouple the distribution and reconstruction of PVSS secrets. Parties continuously produce and distribute new PVSS commitments, which are the encrypted shares and the proofs. Meanwhile, all parties store received commitments using first-in-first-out queues and collectively consume each commitment to recover the corresponding secret for beacon generation. To achieve asynchronous consensus, we employ reliable broadcast for distribution and apply t-validated asynchronous Byzantine agreement for reconstruction. To achieve reconfiguration, honest parties can collectively remove a faulty party if his queue remains empty for an extended duration, and a new party can join the system using reliable broadcast. We also introduce a novel PVSS scheme based on Sigma protocol and Fiat-Shamir heuristic, which is of independent interest. Consequently, AsyRand maintains state-of-the-art complexity with $O(n^2)$ communication complexity, $O(n)$ computation complexity, and $O(n)$ verification complexity while achieving asynchrony and reconfiguration. Experimental results highlight the performance of AsyRand compared to existing works.**

1. Introduction

Distributed randomness beacon protocols are designed to generate trustworthy and verifiable random values continuously [1], [6], [12], [13], [34], [35], [36]. Beacons have a broad spectrum of applications, including secure multiparty computation [2], consensus protocols [6], [18], anonymous communication [7], [8], blockchain sharding [3], [4] and Byzantine agreement protocols [5]. The key properties of a beacon [12], [13], [34] are as follows:

- *Liveness/Availability:* All honest parties continuously proceed to generate new random output for every epoch.
- *Bias resistance:* Each randomness must be unbiased and uniformly distributed.
- *Unpredictability:* No party should be able to predict a beacon's value before it is generated.
- *Public verifiability:* Any party can verify that the beacon values are correctly generated following the protocol.
- *Responsiveness:* Beacon values are delivered at the speed of the actual network.

However, ensuring these properties in an asynchronous network while supporting reconfiguration remains an open challenge [10]. An asynchronous network allows messages to experience arbitrary delays, mirroring real-world network conditions [28], [46], [51]. Reconfiguration [33], [36], [43], the process of updating parties, is essential for removing faulty parties and adding external parties.

1.1. Our Approach in a Nutshell

We adopt publicly verifiable secret sharing (PVSS) (cf. Section 2.1), the producer-consumer model (cf. Section 2.2), reliable broadcast (RBC, cf. Section 2.3) and validated asynchronous Byzantine agreement (t-VABA, cf. Section 2.4) to enable asynchrony and reconfiguration in implementing the proposed AsyRand beacon.

1.1.1. PVSS. We use PVSS as the underlying cryptographic primitive in secrets hiding and revealing, where the secrets serve as the seeds for beacon values generation. In PVSS, a dealer divides a secret into encrypted shares, each paired with a validity proof, forming a commitment that anyone can publicly verify. The dealer distributes the commitment to shareholders, who decrypt and validate their shares using the proofs. A subset of these decrypted shares, which are also publicly verifiable, can reconstruct the dealer's secret.

1.1.2. Producer-Consumer Model. We leverage the producer-consumer model to manage PVSS commitments asynchronously. Each party, playing the role of PVSS dealer, continuously and independently produces PVSS commitments and distributes to all parties. All parties, as shareholders, receive and collectively process these commitments to generate beacon values as follows. Each party locally maintains a collection of first-in-first-out (FIFO) queues $ComQ$, where $ComQ[i]$ stores party i's PVSS commitments. PVSS commitments are publicly verifiable, ensuring that no invalid commitments are added to these queues. Define an epoch as the period in which all parties consume a PVSS commitment. In each epoch e, the objective is to generate a new beacon value R_e. In epoch e, a leader, say party L, is randomly elected using the beacon value R_{e-1} from the previous epoch $e-1$. All parties reconstruct leader

TABLE 1: Comparison of PVSS-based beacon protocols.

Protocol	Network	Liveness	Comm.	Unpred.	Bias-rist.	Comp.	Verif.	Reconfig.	Resp.	Adversary
RandHerd [13]	syn.	✗	$O(c^2 \log n)$	1	✗	$O(c^2 \log n)$	$O(1)$	✗	✗	static
SCRAPE [15]	syn.	✓	$O(n^3)$	1	✓	$O(n^2)$	$O(n^2)$	✗	✗	static
HydRand [12]	syn.	✓	$O(n^2)$	$f+1$	✓	$O(n)$	$O(n)$	✗	✗	static
GRandPiper [33]	syn.	✓	$O(n^2)$	$f+1$	✓	$O(n^2)$	$O(n^2)$	✓	✗	adaptive
SPURT [34]	semi-syn.	✓	$O(n^2)$	1	✓	$O(n)$	$O(n)$	✗	✓	adaptive
OptRand [36]	syn.	✓	$O(n^2)$	1	✓	$O(n)$	$O(n)$	✓	✓	static
GRandLine [35]	syn.	✓	$O(n^2)$	1	✓	$O(n)$	$O(n)$	✗	✓	adaptive
AsyRand	asyn.	✓	$O(n^2)$	$f+1$	✓	$O(n)$	$O(n)$	✓	✓	mobile‡

Unpred. depicts how many future epochs an adaptive rushing adversary can predict. In Hydrand [12], GRandPiper [33] and ours, a worst case exists with little probability where $l(\leq f)$ colluding malicious parties become leaders consecutively. In this occasion, the colluding parties can predict l future beacon values. Randomness is fully unpredictable beyond epoch $e+f$, due to the presence of at least one honest leader in any consecutive $f+1$ epochs.

‡: We enable reconfiguration, meaning parties can join or leave. Thus, we assume a mobile adversary model and $f < n/3$ holds whenever n changes [48].

L's secret gs by consuming the leader's first commitment in $ComQ[L]$. Then, the beacon randomness value R_e of epoch e is calculated as $R_e = Hash(R_{e-1}, gs)$, where $Hash$ is a hash function.

1.1.3. Asynchronous Consensus. To guarantee consistency of PVSS commitments in asynchronous network, we use RBC for producing PVSS commitments and t-VABA for consuming them. Each party, as PVSS dealer, continuously leverages the RBC protocol to achieve consensus on new PVSS commitments among all parties. All parties, as shareholders, only add PVSS commitments that have reached agreement to their local queues. We design a new protocol t-VABA to enable parties, each holding a decrypted share, to collaboratively recover the leader's secret gs in each epoch. Upon completion of t-VABA, the beacon value R_e is successfully delivered.

1.1.4. Achieving Reconfiguration. AsyRand supports reconfiguration by allowing parties to add or remove parties dynamically. On the one hand, honest parties can monitor the status of the queues to judge whether a party is faulty or not. They can automatically trigger removal of party i if $ComQ[i]$ is empty for an extended duration, since party i may have become faulty quietly. On the other hand, a new party θ (outside the system) can be added via proposing his PVSS commitment via a RBC protocol. Once the RBC protocol reaches agreement, each party puts θ's PVSS commitment into its local queue $ComQ[\theta]$.

1.2. Related Works

Many approaches have been explored to build distributed randomness beacon protocols. Heuristically, public random numbers can be obtained as a byproduct of Bitcoin's PoW consensus [11]. Other cryptographic primitives are also employed as the underlying tools of beacon protocols, such as verifiable randomness function (VRF) [9], [18], verifiable delay function (VDF) [22], [37], threshold signatures [19], verifiable secret sharing (VSS) [32], [33], [43] and publicly verifiable secret sharing (PVSS) [12], [15], [34], [35]. Some beacon protocols [19], [20], also necessitate distributed key generation (DKG) [20], [21], [24], [25], [35] during the initial setup. Summarily, these primitives are leveraged to achieve a commit-and-reveal paradigm. In the committing phase, distributed parties introduce private randomness or entropy and broadcast the corresponding commitment to others; In the revealing phase, the random value is uncovered and the final beacon value is calculated.

In the context of constructing a commit-and-reveal paradigm, PVSS schemes [15], [16], [23], [38], [41] are widely adopted as they do not require a private communication channel and can identify faulty parties. RandHerd [13] divides parties into c-size subgroups to achieve scalability with communication and computation complexity $O(c^2 \log n)$ at the cost of higher liveness failure probability. The verification complexity is $O(1)$ due to the use of collective signing with cryptographic multisignatures. In Ouroboros [6] and SCRAPE [15], n PVSS commitments from all parties are published for calculating a beacon value, resulting in $O(n^3)$ communication complexity. Moreover, SCRAPE leverages an optimized PVSS scheme, which reduces its computation and verification complexities to $O(n^2)$. HydRand [12], GRandPiper [33] and AsyRand lower the complexity by randomly choosing leaders in each epoch. In a synchronous network, HydRand and GRandPiper tolerates 1/3 and 1/2 faulty parties, respectively. SPURT [34] improves communication efficiency by aggregating PVSS commitments based on SCRAPE PVSS within a semi-synchronous network. However, SPURT weakens the liveness of the beacon protocol. If the leader of an epoch is faulty, the scheme allows parties to output a bot symbol instead of a randomness value. OptRand [36] further aggregates PVSS commitments and can tolerate up to 1/2 faulty parties, incorporating the advantages of SPURT and GRandPiper. However, OptRand assumes a synchronous network model. GRandLine [35] runs with an optimized DKG in the pre-processing phase, which is based on a recursive aggregable PVSS. Both GRandPiper [33] and OptRand [36] facilitate reconfiguration in a synchronized network. Rondo [43] (based on batched VSS and assuming semi-synchronous network) enables reconfiguration by requiring joining or leaving parties to actively submit online proposals.

Byzantine agreement (BA), Byzantine fault tolerance (BFT) or state machine replication (SMR) [26], [27] are commonly employed to achieve continuous agreement among distributed parties. Thus, the combination of PVSS primitive with BA-, BFT- or SMR-based consensus is widely employed to implement beacon protocols [12], [13], [33], [34], [35], [36].

Table 1 compares the PVSS-based beacon protocols. VSS-based beacon protocols are not included, as they require private communication channels.

1.3. Contributions

• We propose AsyRand, a new PVSS-based beacon for asynchronous network, by leveraging a producer-consumer model, RBC and t-VABA protocols. Each party produces PVSS commitments continuously and independently. Meanwhile, all parties collectively consume each commitment to deliver beacon value in every epoch. The production and consumption of commitments are guaranteed to be consistent via RBC and t-VABA protocols, respectively.

• Our AsyRand protocol supports reconfiguration. An external party can request to join by producing a valid PVSS commitment and distributing it using RBC, and all honest parties will accept this party upon RBC agreement. Each party can detect faulty parties by locally monitoring the state of queues. If the queue corresponding to a party remains empty for an extended duration, all honest parties will trigger the removal of this party via t-VABA protocol.

• We propose t-VABA, a novel Byzantine agreement protocol, where the agreement value may be none of parties' inputs. Instead, the agreement value is collectively determined by any t honest parties. In AsyRand, t-VABA is applied for honest parties to deliver beacon values and to remove faulty parties.

• As of independent interests, we design a new PVSS scheme maintaining state-of-the-art complexity by using Sigma protocol [29] and Fiat-Shamir heuristic [30] to achieve non-interactive zero knowledge (NIZK) proofs. Numerical results show that our PVSS scheme performs well.

2. Preliminaries

2.1. Publicly Verifiable Secret Sharing (PVSS)

A publicly verifiable secret sharing (PVSS) [15], [17], [42] enables a dealer to share secrets among distributed shareholders in a publicly verifiable manner. Particularly, the dealer shares a secret $gs = g^s$ among n shareholders $\mathcal{P} = \{P_1, ..., P_n\}$, where $s \in \mathbb{Z}_p$. A PVSS scheme consists of the following five phases:

1) $(\{\mathsf{sk}_i, \mathsf{pk}_i\}) \leftarrow \mathsf{PVSS.Setup}(\lambda, t, n)$ Each shareholder P_i generates a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
2) $(C, \pi) \leftarrow \mathsf{PVSS.Share}(s, \{\mathsf{pk}_i\})$ The dealer divides the secret into n shares. Each share is encrypted into C_i and all encrypted shares are accompanied by NIZK proofs. Then the dealer publishes the PVSS commitment, i.e., $(C = \{C_i\}, \pi)$.
3) $bool \leftarrow \mathsf{PVSS.Verify}(C, \pi)$ Any external verifier can check whether the dealer has honestly shared a secret given (C, π).
4) $D_i \leftarrow \mathsf{PVSS.PreRecon}(C_i, \mathsf{sk}_i)$ Each shareholder decrypts his encrypted share to obtain decrypted share D_i.
5) $gs \leftarrow \mathsf{PVSS.Recon}(C, \{D_i\}_{i \in T})$ With a set of correct decrypted shares T ($|T| \geq t$), the secret value $gs = g^s$ can be reconstructed. Note that incorrect decrypted shares $\{D_i\}_{i \in T}$ can be detected and be abandoned.

A PVSS scheme satisfies the properties of **correctness**, **public verifiability** and **IND1-Secrecy**.

- ***Correctness*** Correctness ensures that at least t shareholders follow the protocol, the reconstructed secret will be identical to the original secret g^s.
- ***Public Verifiability*** Public verifiability allows anyone to verify the correctness of the encrypted shares. Besides, the decrypted shares should also be publicly verifiable in the reconstruction phase.
- ***IND1-Secrecy*** IND1-Secrecy guarantees that an adversary, given any $t-1$ secret keys $\{\mathsf{sk}_i\}$ and the public information, learns no information about the secret gs.

2.2. Producer-Consumer Model

Producer-consumer model is a coordination pattern where a producer generates data items into a shared buffer while a consumer processes them, enabling asynchronous workflows. The buffer can be implemented using a FIFO queue, which ensures that data is consumed in the same order it is produced. A queue usually has a fixed length. Let $put(\cdot)$ and $get()$ represent the operations for adding data to the queue and retrieving data from the queue, respectively. If each data item is assigned with a sequence number, then items with larger sequence numbers are suspended from being added—even if they are ready earlier than others. We refer the first data item as the earliest unconsumed element in the queue. Figure 1 depicts the producer-consumer model vividly.

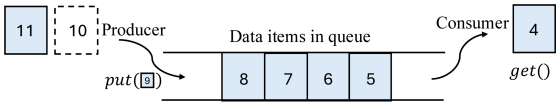


Figure 1: The producer-consumer model

2.3. Reliable Broadcast

Reliable broadcast (RBC) ensures that a message broadcast by an honest dealer is received by all honest parties in a distributed system. Particularly, Bracha RBC [28] is regarded as a one-shot consensus algorithm in an asynchronous setting, tolerating $f < n/3$ faulty parties, where n is the number of total parties. Suppose the proposal is v. We slightly modify the Bracha RBC protocol by replacing

the proposed value v (of size $O(n)$) with its hash value hv. The details of the modified protocol are depicted as below:

Bracha RBC for $O(n)$-size v

Step 0. For the dealer j
broadcasts $(\mathsf{initial}, v)$.

Step 1. For party i, waits until the receipt of $(\mathsf{initial}, v)$
broadcasts (echo, hv).

Step 2. For party i, waits until the receipt of $2f+1$ (echo, hv) or $(f+1)$ (ready, hv)
broadcasts (ready, hv).

Step 3. For party i, accepts v and hv until the receipt of $2f+1$ (ready, hv).

Note that if party i receives $2f+1$ (ready, hv) without v in **Step 3**, it can fetch v from any other parties. The RBC protocol has the following security properties [45], [46]:

- ***Validity*** If the dealer is honest, then all honest parties eventually delivers the dealer's proposal v.
- ***Agreement*** If an honest party delivers a value v', and another honest party delivers v'', then $v' = v''$.
- ***Totality*** If an honest party delivers a value, then all honest parties eventually deliver a value.

It is well-known that RBC protocols cannot handle cases with a faulty dealer [26], as it may either prevent consensus entirely or lead to unbounded termination time in asynchronous settings. We define a symbol "$\perp$", representing an initial consensus status. Also, $\perp$ can be initialized as the default consensus value for all parties. It can be inferred that consensus value can be changed from $\perp$ to some value $v(\neq \perp)$. However, v will be immutable once upon agreement by all honest parties due to **validity**, **agreement** and **totality**. Therefore, Claim 1 holds.

Claim 1. In the Bracha asynchronous RBC protocol, all honest parties eventually share the same consensus value, either $\perp$ or $v(\neq \perp)$.

2.4. t-Validated Asynchronous Byzantine Agreement (t-VABA)

Suppose there are a group of n distributed parties and at most $f < n/3$ of them are faulty. In validated asynchronous Byzantine agreement (VABA) protocols [46], [51], each party has an initial value at the beginning and all honest parties reach agreement on a value in the end. Typically, VABA protocols require that the agreement value is a proposal from a party. In this section, we introduce a t-validated asynchronous Byzantine agreement (t-VABA) protocol.

Definition 1 (t-VABA). Each party i has an initial valid value C_i and all parties' goal is to reach agreement on a valid value v (i.e., $\mathsf{Predicate}(v) = \mathsf{true}$). v is none of the initial values, but determined by any t honest parties, where t is a fixed threshold value.

The predicates $\mathsf{Predicate}$ is utilized to validate the output common value. Similarly to previous VABA protocols [46], [47], t-VABA is required to achieve the following properties:

- ***External-validity***: If an honest party decides on a value v, then v is calculated with any t honest parties' initial values $\{C_i\}$ and $\mathsf{Predicate}(v) = \mathsf{true}$.
- ***Agreement***: All honest parties that terminate decide on the same value v.
- ***Termination***: If honest parties input valid initial values, then they will eventually decide on a value.

3. System and Security Models

3.1. System Model and Assumption

The proposed distributed randomness beacon $\mathsf{AsyRand}$ assumes pairwise connected network for broadcast in an open environment. By "broadcast", we mean the message is sent to all parties (including the sender) through the network. The network model is asynchronous, so messages can be arbitrarily delayed. PKI is also assumed so that each party leverages a public key as its identity.

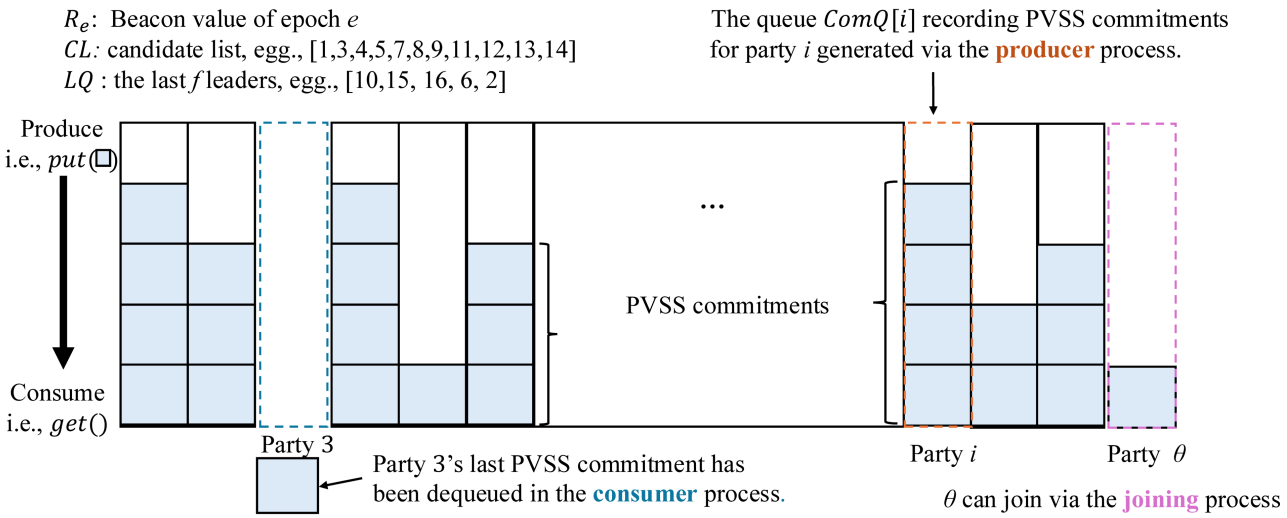
Denote n as the number of parties remaining in the system. f represents the number of faulty parties that may cause Byzantine failures or disobey the protocols. For simplicity, we assume $n = 3f + 1$. Parties can quietly leave without sending any messages. As we allow for parties changing, we assume mobile adversary [48], where the condition $f < n/3$ is assumed to hold regardless of regardless of how n varies.

Each party maintains four processes to produce randomness beacons and handle party joining or removing. Particularly:

- In the producer process, each party continuously produces new PVSS commitments using RBC protocols;
- In the consumer process, a leader is randomly chosen for each epoch and honest parties collectively deliver a beacon value by consuming the leader's first PVSS commitment based on t-VABA protocol;
- In the removal process, honest parties propose to remove faulty parties;
- In the joining process, honest parties collectively decide whether to allow a new party to participate in the system.

We set $t = 2f + 1$ for t-VABA protocols. An epoch is the period of consuming a PVSS commitment (via a t-VABA protocol) in the consumer process. Note that different parties may in different epochs at a specific time due to asynchronous network.

The initial beacon value and the initial leader are assumed to be obtained in a decentralized way, which can be obtained with VDF [37] or a nonce from Bitcoin [11].



Party *3* will be removed in the **removal** process if $|ComQ[3]|$=0 holds long, as the condition halts the **consumer** process.

Figure 2: Local view at a party, depicting the four processes and the queues recording PVSS commitments

3.2. Threat Model

In the system, malicious parties or adversaries can collude to violate the above properties by arbitrarily biasing the protocol. A summary of risks and potential attacks is introduced below: In the producer process, each malicious party has the following options: ① delay the delivery of valid PVSS commitment; ② send invalid PVSS commitment to honest parties; ③ send contradicting valid PVSS commitments to different honest parties; ④ send nothing. In the consumer process, a malicious party i also has similar options in broadcasting a PVSS decryption key D_i in each epoch. Besides, the f malicious parties may collude to predict the random beacon values in advance. We only consider one party joining or removal at a time in the reconfiguration. Actually, simultaneous reconfiguration degrades into sequential, one-by-one reconfiguration.

4. The AsyRand Beacon Protocol

4.1. High-Level Overview and Global States

In the producer process, each party broadcasts new PVSS commitments, which are the outputs of the PVSS.Share algorithm, via continuously invoking RBC protocols. In the consumer process, a leader is randomly selected for each epoch, and the earliest PVSS commitment produced by this leader is consumed to recover a random value with the t-VABA protocol. Further, the recovered random value is adopted to generate a fresh beacon value.

We introduce the global states/variables in AsyRand. The leader queue (LQ) records past f leaders. The candidate list (CL) denotes potential leaders, i.e., the parties that have not been leaders in the past f epochs. This design helps avoid the situation where malicious parties dominate leadership consecutively, giving them an undue advantage in the protocol. It is apparent that $CL = \mathcal{P} - LQ$. $ComQ$ is a collection of FIFO queues. Each PVSS commitment from party i carries a monotonically increasing sequence number j, denoted as seq_{ij}. For notational simplicity, we use seq_i throughout this paper rather than seq_{ij}, as the additional index j serves no meaningful purpose in our writing. Thus, $ComQ[i]$ is a queue, of length *queLen*, recording the PVSS commitments of party i in chronological order. Hence, all parties share the same global states, including $\{\mathsf{pk}_i\}, LQ, CL, R_{e-1}, e, L, ComQ, seq_i, queLen$. Figure 2 depicts all the processes and the global states of AsyRand with concrete examples within a party. Table 2 summarize the global states, which are shared among all processes in a party. Parties operate based on their local states in an asynchronous network, without relying on a global clock.

TABLE 2: The global states shared by all processes

State	description
$\mathcal{P}$	the set of all parties
n, f	the number of parties and faulty parties
e	current epoch
L	the leader of current epoch e
R_{e-1}, R_e	beacon value of epoch $e-1$, e
pk_i	the ith party's public key
LQ	a queue recording past f leaders
CL	the leader candidates list, i.e., $CL = \mathcal{P} - LQ$
$ComQ$	collection of FIFO queues
$ComQ[i]$	the FIFO queue recording unconsumed PVSS commitments from party i
seq_i	the sequence number of a PVSS commitment at party i
queLen	the maximum PVSS commitments each queue can store

4.2. The Producer Process

In the producer process, each party i independently and continuously invokes RBC protocol to broadcast new PVSS

commitments as long as his queue is not full, as depicted by Figure 3. Each of the PVSS commitment is generated using the PVSS scheme inputting a fresh random value $s \in \mathbb{Z}_p$. Particularly, the proposal is set as $v = (i, C, \pi, seq_i)$ in **Step 0**, where (C, π) is output of the $\mathsf{PVSS.Share}(s, \{\mathsf{pk}_i\})$ algorithm and seq_i denotes the seq_i-th PVSS commitment. The public verifiability property of PVSS scheme enables anyone to check whether an RBC dealer has honestly produced PVSS commitments by leveraging the PVSS.Verify algorithm in **Step 1**. Once an RBC instance is accomplished in **Step 3**, each party updates the global state by $ComQ[v.i].put(v)$.

Step 0. For party i, waits until $|ComQ[i]| \neq queLen$, invokes $(C, \pi) \leftarrow \mathsf{PVSS.Share}$. Denotes $v = (i, C, \pi, seq_i)$ and broadcasts $(\mathsf{initial}, v)$.

Step 1. For party i, waits until the receipt of $(\mathsf{initial}, v)$, where $\mathsf{PVSS.Verify}(v.C, v.\pi)$ is true broadcasts (echo, hv), where $hv = Hash(v)$.

Step 2. For party i, waits until the receipt of $2f + 1$ (echo, hv) or $f + 1$ (ready, hv) broadcasts (ready, hv).

Step 3. For party i, waits until the receipt of $2f + 1$ (ready, hv) $ComQ[v.i].put(v)$.

Figure 3: The producer process at each party i

Claim 2 ($ComQ[i]$ is in order). The values in $ComQ[i]_{\forall i}$ are in the right order for all honest parties.

$Proof$. We leverage seq_i to mark the seq_i-th PVSS commitment of party i. Therefore, it is impossible for an honest party to accept two contradicting PVSS commitments for a sequence number seq_i, due to the RBC **agreement** property. So, values in $ComQ[i]_{\forall i}$ are in the right order for all honest parties.

In an asynchronous RBC protocol, the absence of network timeout assumptions prevents honest parties from reliably determining whether a dealer i is faulty or not. By Claim 1, all honest parties will eventually have the same consensus value $\perp$ or v. If they agree on v, they will append v to $ComQ[i]$ as the seq_i-th item; otherwise, they will take no action to the queue $ComQ[i]$. Thus, honest parties can shift their focus to the status of the queue $ComQ[i]$ and Claim 3 holds.

Claim 3 (Focus on queues). Honest parties in the producer can just focus on the queue status $ComQ[i]$ when party i is the dealer, instead of accessing i's honesty through message analysis. Thus, the potential threats in the producer process defined in system model (cf. Section 3.2) are addressed.

We prove the safety and liveness for the producer process by Theorem 1 and Theorem 2.

Theorem 1 (Safety of the producer process). No matter whether a dealer i is faulty or not, the result of the producer process, $ComQ[i]$, will reach agreement for all honest parties.

$Proof$. The producer process consists of infinite RBC instances invoked by each party. Any two RBC instances led by two dealers are independent. Hence, we discuss the case where any party i invokes the RBC protocols as the dealer. By Claim 3, honest parties can ignore whether a dealer i is honest or not. Moreover, the elements in $ComQ[i]$ are in right order by Claim 2. Therefore, the global state $ComQ[i]$ will be eventually under consensus for all honest parties.

Theorem 2 (Liveness of the producer process). No adversary could prevent honest parties from putting valid PVSS commitment into $ComQ[i]$ in the producer.

$Proof$. The producer process consists of infinite independent RBC protocols led by all parties separately. The proposal (i.e., PVSS commitment) of party i does not interfere with the proposal of another party j, since they have independent storage space, i.e., $ComQ[i]$ and $ComQ[j]$. Since $n = 3f + 1$, the $2f + 1$ honest parties continuously propose new valid PVSS commitments, which will reach agreement among them due to RBC **validity**, guaranteeing the liveness of the producers.

As each PVSS commitment has a sequence number and each party independently produces its own PVSS commitments. The back pressure problem is avoided since honest parties will wait if their queues are full. A party's PVSS commitments can be discarded when $ComQ[i]$ is full for honest receivers. However, rational parties will avoid this behavior, as it would result in their detection as faulty and being eliminated via the removal process. Each party can broadcast multiple commitments with a single RBC protocol. We will demonstrate the performance improvement in experimental results.

4.3. The Consumer Process

The consumer process continuously choose a random leader L for each epoch e. In each epoch, all parties cooperate to recover L's first PVSS commitment and reach consensus on a beacon value. Figure 4 depicts the consumer process at party i for some epoch e using a t-VABA protocol (cf. Definition 1). The t-VABA protocol begins with each party holding an initial value C_i (in C) and it reaches agreement on the beacon value R_e in the end. The implementation resembles RBC protocol but without a dealer in **Step 0**.

The gray text in Figure 4 considers the situation whether the leader L is removed or not. The removal process is introduced in Section 4.4. The FLP impossibility theorem [49] proves that no deterministic asynchronous consensus protocol can guarantee both safety and liveness when even one process may fail. To resolve the problem whether L is truly removed by all honest parties, asynchronous binary agreement (ABA) protocol [50] is integrated into the

consumer process. ABA is widely adopted to achieve one-shot byzantine consensus, where each party inputs a bit 0 or 1, in asynchronous environment. The ABA protocol immediately outputs 0 (indicating $L.status \neq removed$) if $|ComQ[L]| > 1$ is detected in **Step 0**.

Step 0. For party i, $L = CL[R_{e-1} \ mod \ |CL|]$ and $v \leftarrow ComQ[L].get()$, parse v as (L, C, π, seq_L). Waits until $|ComQ[L]| > 0$ or $L.status \equiv removed$:
$oldL \leftarrow LQ.get()$ and $CL.add(oldL)$
$D_i \leftarrow \mathsf{PVSS.PreRecon}(C_i, \mathsf{sk}_i)$
(Optional) starts ABA protocol to decide $L.status$
broadcasts (recon, D_i).

Step 1. For party i, waits until the receipt of t $(\mathsf{recon}, D_i, *)$,
$gs \leftarrow \mathsf{PVSS.Recon}(C, \{D_i\})$
$R_e \leftarrow Hash(R_{e-1}, gs)$
broadcasts $(\mathsf{reconEcho}, e, R_e)$.

Step 2. For party i, waits until the receipt of $2f+1$ $(\mathsf{reconEcho}, e, R_e)$ or $f+1$ $(\mathsf{reconReady}, e, R_e)$
broadcasts $(\mathsf{reconReady}, e, R_e)$.

Step 3. For party i, waits until the receipt of $2f+1$ $(\mathsf{reconReady}, e, R_e)$,
accepts R_e
(Optional) waits ABA termination of $L.status$, and if $L.status = removed$:
$LQ.put(L)$ and $CL.remove(L)$
enters **Step 0** to the next epoch $e+1$.

Figure 4: The consumer process leveraging a t-VABA

The t-VABA protocol is implemented following a four-step structure analogous to the RBC protocol. In **Step 0**, each party determines current leader as $L = CL[R_{e-1} \ mod \ |CL|]$, where R_{e-1} is the beacon value of previous epoch $e-1$. The party then obtains L's first PVSS commitment by $v \leftarrow ComQ[L].get()$, and parse v as (L, C, π, seq_L). The waiting condition $|ComQ[L]| > 0$ means that the t-VABA protocol really starts only if L has 1 or more PVSS commitments to consume. This design proactively prevents the case where L is re-elected as leader without having an unused PVSS commitment available, thus eliminating potential liveness stalls in the consumer process.

If the above condition is met, the oldest leader $oldL$ (i.e., $LQ.get()$) should be placed into candidate list CL. And if L is still alive in the system, it should be transferred from the candidate list CL to the leader queue LQ. The encrypted share C_i (in C) serves as party i's initial value. Subsequently, each party i obtains the decryption key $D_i \leftarrow \mathsf{PVSS.PreRecon}(C_i, \mathsf{sk}_i)$ for the commitment v and broadcast it. In **Step 1**, each party recovers the PVSS secret gs with at least t valid decryption keys, and constructs the value R_e. The **Step 2** and **Step 3** are the same with the RBC protocol (cf. Section 2.3). In the end, all honest parties accepts R_e as the beacon value of the epoch e.

In summary, the consumer process updates the global variables, i.e., L, CL, LQ, $ComQ$ and outputs new beacon value R_e for each epoch e. Obviously, the execution of the consumer process relies on the producer process. The consumer process should be "slower" than the producer process, so that the operation $ComQ[L].get()$ always returns a valid PVSS commitment in every epoch. The fact is guaranteed and it is proved by Claim 4.

Claim 4 ($ComQ[i]$ is non-empty). The set $ComQ[i]$ is non-empty, ensuring that the consumer process always has a valid PVSS commitment to consume whenever a party i is chosen as leader.

Proof. By Theorem 2, the queue $ComQ[i]$ can be appended without interruption. The condition, i.e., $|ComQ[i]| > 0$, in **Step 0** guarantees that $ComQ[i]$ is non-empty for all honest parties. If party i stops producing valid PVSS commitments in the producer process, $ComQ[i]$ will stop growing. $ComQ[i]$ will be empty in **Step 0** if i is chosen as leader for $|ComQ[i]|$ times. In this case, the consumer process will be suspended and it will be resumed after removing i by the removal process.

We argue that the t-VABA implementation satisfies the required security properties through Lemma 1, Lemma 2, and Lemma 3, respectively. Further, we prove the consumer process ensures properties of safety and liveness by Theorem 3 and Theorem 4.

Claim 5 (Same beginning). At the beginning of each epoch e with leader L, all honest parties share the same value of $v \leftarrow ComQ[L].get()$ in **Step 0**.

Proof. In **Step 1** of each t-VABA protocol, R_e is deterministic due to correctness of PVSS scheme in computing gs and uniqueness of hash in calculating R_e. Thus, it can be easily proved that honest parties have the same output at the beginning of each epoch by applying mathematical induction. Moreover, the producer process guarantees that all honest parties have correct order of $ComQ[L]$ (cf. Claim 2). And there is at least one PVSS commitment in the queue $ComQ[L]$ (cf. Claim 4). Therefore, all honest parties have the same value of $ComQ[L].get()$ at the beginning of each epoch.

Lemma 1 (External-validity of each epoch). If an honest party outputs a beacon value R_e for epoch e, then R_e is calculated with any t honest initial values $\{C_i\}$ and $\mathsf{Predicate}(R_e) = \mathsf{true}$.

Proof. By Claim 5, honest parties have the same PVSS commitment to consume at each epoch. Specifically, the initial value for each party is $C_i \leftarrow v.C_i$, where $v \leftarrow ComQ[L].get()$ and it is parsed as $(L, C = \{C_i\}, \pi, seq_L)$. Moreover, C_i has been validated by the $\mathsf{PVSS.Verify}$ algorithm in the producer process. Then, each honest party i will decrypt C_i to obtain D_i via $\mathsf{PVSS.PreRecon}$ and broadcast (recon, D_i) in **Step 0**. Each party can collect at least $n - f > t$ valid $\{D_i\}$, enabling it to calculate gs by $\mathsf{PVSS.Recon}(C, \{D_i\})$. Further, the beacon value R_e is

calculated as $R_e = Hash(R_{e-1}, gs)$. All honest parties will send $(\mathsf{reconEcho}, e, R_e)$ in **Step 1**. The subsequent steps (i.e., **Step 2** and **Step 3**) are the same with RBC protocol, guaranteeing that all honest parties will receive $2f+1$ reconReady messages and accept the same beacon value R_e as the beacon value for epoch e. Therefore, the Predicate roughly consists of PVSS.PreRecon, PVSS.Recon and $Hash$ algorithms.

Claim 6. No two honest parties will send conflicting messages $(\mathsf{reconReady}, e, R_e)$ and $(\mathsf{reconReady}, e, R'_e \neq R_e)$, given the correctly chosen leader L at epoch e.

$Proof$. Claim 5 has demonstrated that honest parties are consuming the same commitment for epoch e. Suppose two honest parties i and j send reconReady message for two different beacon values R_e and R'_e, respectively. Party i must have received a set A of $2f+1$ reconEcho for R_e and party j must have received a set B of $2f+1$ reconEcho for R'_e. Since $|A| = |B| = 2f+1$, then $|A \bigcap B| \geq f+1$ due to quorum intersection property. This implies that at least $f+1$ parties have sent an echo to both i and j. It means that at least one honest party must have sent two reconReady messages for different values, violating the assumption.

Lemma 2 (Agreement of each epoch). If an honest party accepts R_e, then all honest parties accept R_e for epoch e.

$Proof$. First consider the scenario where faulty parties collude or send invalid decryption keys $\{D_i\}$ for current leader's PVSS commitment in **Step 0**. However, invalid keys can be detected by the $\mathsf{PVSS.Recon}(C, \{D_i\})$ algorithm and they will be abandoned. And the PVSS threshold $t = 2f+1 > f$, making it impossible for colluding parties to recover the leader L's PVSS secret gs. Further, Claim 6 shows that no contradicting beacon values can be output for two honest parties. Hence, all honest parties will reach agreement on the same beacon value R_e for epoch e.

Lemma 3 (Termination of each epoch). The consumer process eventually outputs a valid beacon value R_e for the epoch e.

$Proof$. By Claim 4, $ComQ[i]$ is always non-empty for each epoch. When a party i is elected as leader, **Step 0** can be guaranteed to be executed for all honest parties. And honest parties have the same PVSS commitment to consume by Claim 5. The subsequent steps **Step 1-3** follow the same structure as the RBC protocol. Thus, all honest parties will eventually accept R_e for epoch e, similar to RBC **validity**. Hence, each epoch achieves the ***termination*** property of t-VABA protocol.

Theorem 3 (Safety of the consumer process). All honest parties output the same beacon value in each epoch.

$Proof$. The safety property is implied by the ***agreement*** of each epoch (cf. Lemma 2). Thus, the consumer process will always reach agreement for all honest parties for any epoch.

Theorem 4 (Liveness of the consumer process). No adversaries could prevent the consumer process from outputting a new beacon value and forwarding to the next epoch.

$Proof$. The t-VABA ***termination*** property implies that each epoch eventually ends. Besides, the consumer process can successfully move to the next epoch due to Claim 4. Thus, the consumer process normally proceeds with infinite t-VABA instances. We further consider an abnormal case, where the **Step 0** may suspend due to $|ComQ[L]| = 0$ holds. However, in this occasion, party L will be removed in the removal process and it will not be elected as leader. The removal process notifies the consumer process to go on from the suspended point in **Step 0**. Moreover, ABA protocol is leveraged to eliminate the possibility of forks, induced by faulty parties, which may impact liveness. Therefore, the consumer process is guaranteed to achieve liveness.

4.4. The Removal Process

The removal process is designed to enable honest parties to automatically remove faulty members without system restart. Implied by Claim 3, we innovatively invent a mechanism to discover faulty parties in an asynchronous network. The mechanism allows a party to simply detect whether another party is faulty/malicious by monitoring queue length of the party. If the queue is empty for a long time, honest parties can start to remove the party using a t-VABA protocol.

Step 0. For party i, if $|ComQ[L]| = 0$ holds for an expiration event. Denote $v = (L, e)$,
broadcasts $(\mathsf{removal}, v)$.

Step 1. For party i, waits until the receipt of $2f+1$ $(\mathsf{removal}, v)$,
broadcasts $(\mathsf{removalEcho}, v)$.

Step 2. For party i, waits until the receipt of $2f+1$ $(\mathsf{removalEcho}, v)$ or $f+1$ $(\mathsf{removalReady}, v)$
broadcasts $(\mathsf{removalReady}, v)$.

Step 3. For party i, waits until the receipt of $2f+1$ $(\mathsf{removalReady}, v)$,
set *L.status = removed*
notify the consumer process.

Figure 5: The removal process to remove L at party i

For example, a party L may behave maliciously (the possible behaviors are introduced in Section 3.2) in the producer process, and no PVSS commitments will be added into $ComQ[L]$ for all honest parties. $ComQ[L]$ will eventually become empty after L is elected as leader for $|ComQ[L]|$ times. In this case, the leader L has no unconsumed commitments, i.e., $|ComQ[L]| = 0$ in the consumer process (cf. Figure 4) until an expiration event[1], honest party i

1. In our design, the expiration event is defined as the condition where at least $2f+1$ queues contain two PVSS commitments. This threshold aligns with the mobile adversary model's core assumption—that at least two-thirds of the remaining participants are honest.

broadcasts the initialization message $(\mathsf{removal}, v)$, where $v = (L, e)$, in **Step 0**. The following steps are similar to the RBC protocol. Once a party i receives $2f + 1$ $(\mathsf{removalReady}, v)$, it marks that L is removed and notifies its consumer process, which was suspended in **Step 0** in consumer process. Figure 5 depicts the party removal process in a t-VABA protocol. We assume that messages in the removal process are well-signed.

We prove the removal process satisfy the required ***external-validity***, ***agreement*** and ***termination*** properties of t-VABA protocol by Lemma 4, Lemma 5 and Lemma 6, respectively.

Lemma 4 (External-validity of the removal process). If an honest party agrees to remove L which is faulty in epoch e, then $(\mathsf{removal}, v = (L, e))$ has been proposed by at least t parties and $\mathsf{Predicate}(v) = \mathsf{true}$.

$Proof.$ If an honest party agrees to remove L, then at least $2f + 1$ parties have broadcast $\mathsf{removalReady}$ in **Step 3**. Further, it can be inferred that at least $2f + 1 = t$ parties have broadcast $\mathsf{removal}$ in **Step 1**. The $\mathsf{Predicate}$ algorithm can be obtained by collecting the messages of the t parties.

Claim 7. No two honest parties will send conflicting message $(\mathsf{removalReady}, v)$ and $(\mathsf{removalReady}, v' \neq v)$, where $v.e = v'.e$.

$Proof.$ That means given a specific epoch e, no removal agreement will achieve to remove multiple leaders. The proof follows the same reasoning as in Claim 6 and is therefore omitted here.

Lemma 5 (Agreement of the removal process). If an honest party set $L.status = removed$ in epoch e, then eventually all honest parties will agree to set $L.status = removed$.

$Proof.$ If an honest honest party remove party L, it must have received $2f + 1$ $(\mathsf{removalReady}, v)$ messages, of which at least $f+1$ came from honest parties. Consequently, all honest parties will broadcast $(\mathsf{removalReady}, v)$, either due to seeing the $f + 1$ $\mathsf{removalReady}$ messages or due to seeing $2f + 1$ $\mathsf{removalEcho}$, as described in **Step 2**. Claim 7 guarantees that no honest parties will broadcast $(\mathsf{removalReady}, v' \neq v)$. So there will not be $2f + 1$ $\mathsf{removalEcho}$ for v' or $f+1$ $\mathsf{removalReady}$ for v'. Hence, all honest parties will agree to set $L.status = removed$.

Lemma 6 (Termination of the removal process). All honest parties will eventually terminate with agreement on whether $L.status = removed$ of epoch e.

$Proof.$ It is apparent that all honest parties will terminate if at least $2f + 1$ parties incur expiration event. Otherwise, honest parties may halt in the removal process. Due to the use of ABA protocol in the consumer process, honest parties can terminate the removal process upon transitioning to new epochs, where $L.status$ of epoch e is determined.

4.5. The Joining Process

We further introduce a party joining process that enables the seamless addition of a new party without system restart.

Suppose a joining party θ composes a proposal $v = (e, e^*, \mathsf{pk}_\theta, C, \pi, seq_\theta = 1)$, where e is the current epoch, $e^* \gg e$ is the expected epoch for θ to appear in the system, pk_θ is θ's public key, (C, π) is the output of $\mathsf{PVSS.Share}(s, \{\mathsf{pk}_i\} \cup \mathsf{pk}_\theta)$ and seq_θ is initialized to be 1 as the first proposal. Then, θ initiates the RBC protocol to request to join the system by broadcasting (join, v). Figure 6 depicts the party joining process using RBC protocol.

Step 0. For a joining party θ, invokes $(C, \pi) \leftarrow \mathsf{PVSS.Share}$. Set $v = (e, e^*, \mathsf{pk}_\theta, C, \pi, seq_\theta = 1)$:
broadcasts (join, v).

Step 1. For party i, waits until the receipt of (join, v), where $\mathsf{PVSS.Verify}(v.C, v.\pi)$ is true
broadcasts $(\mathsf{joinEcho}, hv)$, where hv is hash of v.

Step 2. For party i, waits until the receipt of $2f + 1$ $(\mathsf{joinEcho}, hv)$ or $f + 1$ $(\mathsf{joinReady}, hv)$
broadcasts $(\mathsf{joinReady}, hv)$.

Step 3. For party i, waits until the receipt of $2f + 1$ $(\mathsf{joinReady}, hv)$ and it does not exceed e^*:
inputs 1 for ABA protocol:

Step 4. For party i, waits until e^*,
starts ABA and if it outputs 1:
$ComQ[\theta].put(v)$
$CL.add(\theta)$

Figure 6: The joining process for party θ at party i

The **agreement** of RBC protocol guarantees honest parties to reach consensus in **Step 3**. However, it does not guarantee that all honest parties have entered **Step 3** before epoch e^*. We leverage ABA protocol in **Step 4** to ensure that all honest parties reach consensus on admitting the joining party θ at epoch e^*. Only if ABA outputs 1, honest parties put the joining party θ in CL and place the proposal v in $ComQ[\theta]$. This design avoids forks in the joining process. Clearly, the ABA protocol outputs 1 immediately when the joining party is honest and has stable network connectivity.

In our design, honest parties refuse other joining requests within epoch e^*, thus, we can only consider the one-by-one sequential joining in this paper. We adopt a sequential joining approach to preserve committee stability, as this guarantees the consistent participation required for RBC consensus. Our paper does not consider the placeholder attack—an attack scenario wherein a joining party deliberately blocks other nodes from joining before epoch e^*. We posit that, in practical deployments, this vulnerability can be mitigated by incorporating an effective incentive mechanism or staking protocol.

4.6. Summary of the Processes

Bracha Reliable Broadcast (RBC) or t-VABA protocol is a one-shot communication abstraction to achieve consensus. To provide a concise overview of the processes, Table 3 is presented. The column **Instance(s)** indicates how many RBC or t-VABA protocols are executed in each process. The column **Actions** summarizes the PVSS algorithms involved in each process. The column **States** introduces the global states which are updated in each process.

TABLE 3: Summary of the processes in AsyRand

Process	Protocol	Instance(s)	Actions	States
Producer	RBC	infinite	PVSS.Share PVSS.Verify	$ComQ[L]$
Consumer	t-VABA	infinite	PVSS.PreRecon PVSS.Recon	LQ, CL, L $ComQ[L], e, R_e$
Removal	t-VABA	1	–	L
Joining	RBC	1	PVSS.Share PVSS.Verify	$CL, ComQ[\theta]$

The producer and joining processes are directly leverage the RBC protocol. All parties first invoke the PVSS.Share algorithm and initiate the RBC protocol with the output PVSS commitment as proposal. Subsequently, PVSS.Verify is triggered when a party receives the initial or join message.

The consumer and removal processes are constructed based on t-VABA protocols. In the consumer process, the PVSS.PreRecon and PVSS.Recon algorithms are executed to recover the leader's secret. Both the producer and consumer processes run with an infinite number of RBC and t-VABA instances, respectively, which form the foundation of AsyRand. The removal and joining processes require only a single instance of the RBC and t-VABA protocol, respectively.

The RBC protocol has communication complexity of $O(n^2)$ in the producer process, where parties broadcast $O(n)$-size PVSS commitments. In the consumer process, each party only broadcasts $O(1)$ size messages in each step. The ABA protocol, which is optionally included in the consumer process, costs $O(n^2)$ communication complexity. Consequently, the complexity of the consumer process is also $O(n^2)$. Therefore, the overall communication complexity of AsyRand is $O(n^2)$.

5. Properties of AsyRand

Theorem 5 (Liveness/Availability). No adversaries could prevent the processes in AsyRand from proceeding.

$Proof$. We discuss all the four processes in AsyRand described in Section 4. By Theorem 2, we prove that no adversaries can prevent new PVSS commitments from being appended into global state $ComQ$ for honesty parties. By Theorem 4, we prove that no adversaries could prevent a PVSS commitment from being consumed. The removal process and the joining process is guaranteed to achieve liveness, due to the t-VABA **termination** and RBC **validity**, respectively. Further, the PVSS commitment queue $ComQ[L]$ of the leader L is always non-empty in the consumer process by Claim 4. Then, a secret gs hidden in $v \leftarrow ComQ[L].get()$ can be eventually recovered with $2f + 1$ $(= t)$ PVSS decrypted shares from honest parties. Therefore, a new beacon value R_e for epoch e is guaranteed to be calculated by $Hash(gs, R_{e-1})$, showing property of guaranteed output delivery [10], [12].

Theorem 6 (Bias Resistance). Adversaries cannot bias the beacon output R_e of epoch e in a predictable way.

$Proof$. As designed, beacon values are delivered in the consumer process. In the consumer process, the condition $|ComQ[L]| > 0$ holds (cf. Figure 4) for leader L when the earliest PVSS commitment $v \leftarrow ComQ[L].get()$ is consumed. Moreover, the leader queue LQ guarantees that a party can be chosen as leader again only after f epochs. Hence, the random beacon value of epoch e, $R_e = Hash(gs, R_{e-1})$, depends on the secret gs hidden in a PVSS commitment produced at least f epochs earlier. Further, R_{e-1}, which also affects the value of R_e, is available at the end of epoch $e-1$. Summarily, gs and R_{e-1} are determined by past commitments and cannot be controlled or predicted by adversaries in the current epoch e. Thus, it is impossible for adversaries to bias the beacon output R_e in a meaningful manner.

Theorem 7 (Unpredictability). An adversary should not be able to predict (precompute) a future beacon value R_{e+f+1}, where e is the epoch of current time.

$Proof$. As described in Theorem 6, predicting a future random beacon value R_e requires knowledge of both PVSS secret gs and the previous beacon value R_{e-1}. PVSS ***IND1-Secrecy*** ensures that the PVSS secret remains indistinguishable until it is reconstructed for honest parties. The PVSS secret can be recovered only when at least $t(> f)$ distinct (recon, D_i) messages are broadcast. Consider the worst-case scenario where the f colluding malicious parties share their PVSS secrets privately in real time. If $l \leq f$ of these malicious parties are selected as leaders in consecutive epochs starting from epoch e, then the beacon values from epoch e to epoch $e + l$ can be calculated in advance. The probability of this case can be modeled as hypergeometric distribution [12]. A party will be elected as leader at least f epochs later, making it impossible to predict beacon value after epoch R_{e+f+1}. Therefore, to ensure complete unpredictability in practice, it is recommended to use future beacon values beyond epoch $e + f$.

Theorem 8 (Public Verifiability). Any third party with publicly known information can verify the correctness of beacon value R_e of each epoch e.

$Proof$. The AsyRand public verification property essentially inherits from the PVSS functionality. In the producer process, PVSS commitments are generated using the PVSS.Share algorithm and broadcast in an open network, making them publicly verifiable with the PVSS.Verify algorithm. In the consumer process, the decrypted PVSS shares, obtained using the PVSS.PreRecon algorithm, are also broadcast in open network. These shares are then further verified with the PVSS.Recon algorithm before being used

to recover the PVSS secret. Both the PVSS algorithms and the calculation of beacon value are deterministic, enabling any third party to verify the entire process of generating R_e.

Theorem 9 (Responsiveness). AsyRand is responsive, meaning that beacon values are delivered at the speed of the real network.

Proof. By Claim 4, there always exists a PVSS commitment to consume in each epoch. Thus, beacon values can be delivered as the consumer process proceeds. From a resource competition perspective, the consumer process is typically slower than the producer. This is because while RBC instances (in the producer) run in parallel across all parties, t-VABA execution (in the consumer) requires joint participation. Theoretically, n PVSS commitments can reach consensus within the time needed to output one beacon value. To balance this, our implementation strategically throttles the producer to optimize consumer throughput. Therefore, AsyRand delivers beacon values at the pace of the consumer process, advancing at the speed of the actual asynchronous network.

6. New PVSS Construction

6.1. Rationale

Previous studies [15], [23], [38] have shown that Shamir secret sharing (cf. Section B) is a primary ingredient to implement PVSS schemes. To ensure that the secret shares can be publicly transferred, each share $p(i)$ should be properly encrypted, where $p(x)$ is a polynomial and $s = p(0)$ is the secret. A natural intuition is to hide the ith share with the ith shareholder's public key pk_i, i.e., $\mathsf{pk}_i^{p(i)}$. The dealer should then prove honesty in the hiding, enabling the encrypted shares to be publicly verifiable.

SCRAPE [15] is notable for being the first PVSS scheme with $O(n)$ verification complexity. SCRAPE chooses another generator $h \in \mathbb{G}$ and computes $h^{p(i)}$ to validate the dealer's honesty through pairing or DLEQ proof [39]. Moreover, the scheme ensures the n shares $p(i)$ correctly interpolate the secret s leveraging Reed-Solomon codes [40]. Similar to SCRAPE, more PVSS schemes with $O(n)$ verification complexity (ALBATROSS [16], HEPVSS/DHPVSS [17]) are proposed.

Sigma protocol is a practical approach to generate zero knowledge proofs, which can be made non-interactive using the Fiat-Shamir heuristic (cf. Section A). Thus, the dealer can prove knowledge of each share $p(i)$ by publishing the statement $\mathsf{pk}_i^{p(i)}$, leveraging NIZK proofs from the Sigma protocol. Particularly, the dealer generates NIZK proofs for each statement $\{\mathsf{pk}_i^{p(i)}\}$ for all $i \in [1, n]$. Moreover, the NIZK proofs should also enable to prove that the n witnesses $\{p(i)\}_{i\in[1,\cdots,n]}$ correspond to the unique secret s. In our solution, we establish this correspondence by leveraging Lagrange interpolation.

6.2. Construction

Figure 7 depicts the concrete construction of the proposed PVSS scheme using Sigma protocol and NIZK proof. The required security properties are proved in Appendix C.

Functionality The proposed PVSS scheme

$\underline{(\{\mathsf{sk}_i, \mathsf{pk}_i\}) \leftarrow \mathsf{PVSS.Setup}(\lambda, t, n) :}$

$g \in \mathbb{G}$, $\mathsf{sk}_i \xleftarrow{R} \mathbb{Z}_p$, $\mathsf{pk}_i \leftarrow g^{\mathsf{sk}_i}$

$\underline{(C, \pi) \leftarrow \mathsf{PVSS.Share}(s, \{\mathsf{pk}_i\}) :}$

$p(x) \xleftarrow{R} poly(\cdot)$, where $p(0) = s$

$C = \left\{\{C_i = \mathsf{pk}_i^{p(i)}\}_{i\in[1,\cdots,n]}\right\}$

$p'(x) \xleftarrow{R} p(x)$,where $p'(0) = s' \xleftarrow{R} \mathbb{Z}_p$

$C' = \left\{\{C'_i = \mathsf{pk}_i^{p'(i)}\}_{i\in[1,\cdots,n]}\right\}$

$$\pi \leftarrow \begin{cases} C', \\ c = H(C, C'), \\ \tilde{s} = s' - cs, \\ \{\tilde{p}(i) = p'(i) - c \cdot p(i)\}_{i\in[1,\cdots,n]}, \end{cases}$$

$\underline{bool \leftarrow \mathsf{PVSS.Verify}(C, \pi) :}$

$$\begin{cases} \{C'_i \stackrel{?}{=} \mathsf{pk}_i^{\tilde{p}(i)} \cdot C_i^c\}_{i\in[1,\cdots,n]}, \\ \tilde{s} \stackrel{?}{=} \mathsf{intpl}(\{(i, \tilde{p}(i))\}_{i\in[1,\cdots,n]}) \end{cases}$$

$\underline{D_i \leftarrow \mathsf{PVSS.PreRecon}(C_i, \mathsf{sk}_i) :}$

$D_i = C_i^{1/\mathsf{sk}_i} = g^{p(i)}$

$\underline{gs \leftarrow \mathsf{PVSS.Recon}(C, \{D_i\}_{i\in T}) :}$

$e(D_i, \mathsf{pk}_i) \stackrel{?}{=} e(g, C_i)\ \forall i \in T$

$\mu_i = \prod_{j\in T, j\neq i} \frac{j}{j-i}$

$gs \leftarrow \prod_{i\in T} D_i^{\mu_i} = \prod_{i\in T} g^{\mu_i \cdot p(i)} = g^{\sum_{i\in T} \mu_i \cdot p(i)} = g^s$

Figure 7: Construction of the proposed PVSS scheme

The PVSS.Setup algorithm takes the security parameter λ, n, t as the input and generates an independent generator g of $\mathbb{G}$. In this algorithm, each shareholder P_i generates a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and the dealer collects all the public keys $\{\mathsf{pk}_i\}$.

In PVSS.Share, the dealer inputs a secret $s \xleftarrow{R} \mathbb{Z}_p$ and all shareholders' public keys $\{\mathsf{pk}_i\}$. The dealer chooses a random polynomial $p(x)$ with the condition $p(0) = s$. Calculate each encrypted share $p(i)$ as $C_i = \mathsf{pk}_i^{p(i)}$. Denote $C = \{\{C_i\}_{i\in[1,\cdots,n]}\}$. Further, the dealer generates the Sigma protocol transcript, i.e., the Sigma commitment C', the challenge c, the response $\tilde{s}$ and $\tilde{p}(i)$. Denote $\pi = (C', c, (\tilde{s}, \{\tilde{p}(i)\}))$ as the NIZK proof. Finally, the dealer publishes the PVSS commitment, i.e., (C, π) publicly.

With PVSS.Verify, anyone can check the validity of the commitment (C, π). In the algorithm, each encrypted share C_i is verified separately and the Lagrange interpolation is leveraged to verify that all shares are generated from the secret s. Particularly, $\mathsf{intpl}(\{(i, \tilde{p}(i))\}_{i\in[1,\cdots,n]})$ recovers the

secret of polynomial $\tilde{p}(x)$, i.e., $\tilde{p}(0)$, where $\tilde{p}(x) = p'(x) - c \cdot p(x)$. That is correct due to additive homomorphism of polynomials.

In PVSS.PreRecon, each shareholder P_i privately decrypts the share $D_i = C_i^{1/\mathsf{sk}_i}$ and sends D_i to a recoverer.

In PVSS.Recon, the recoverer firstly checks validity of D_i from each shareholder. Denote T as the indexes of t valid decrypted shares. Then, the recoverer constructs the dealer's secret $gs = g^s$, in which Lagrange interpolation is implied.

6.3. Complexity Comparisons

Computation Complexity: In the PVSS.Share phase, the dealer costs n exponentiations to generate C. And the NIZK proofs, generated from Sigma protocol, $\pi = (C', c, \tilde{s}, \{\tilde{p_i}\}_{i\in n})$, where C' also takes n exponentiations. Hence, the PVSS.Share phase takes $2n$ exponentiations. In the PVSS.Verify phase, it costs 2 exponentiations for verifying each C_i. Therefore, the PVSS.Verify phase takes $2n$ exponentiations. In the PVSS.Recon phase, it costs $2t$ pairings to check validity of $\{D_i\}$ and t exponentiations to calculate gs.

Communication Complexity: In the sharing phase, the dealer publishes the encrypted shares C and the corresponding NIZK proofs $\pi = (C', c, \tilde{s}, \{\tilde{p_i}\}_{i\in n})$. C contains n elements in $\mathbb{G}$, and the NIZK proofs π contain n elements in $\mathbb{G}$ and $n+2$ elements in $\mathbb{Z}_p$. In the reconstruction phase, the recoverer receives an array $\{D_i\}_{i\in T}$ to decrypt C. Hence, reconstruction phase requires t elements in $\mathbb{G}$ to recover the secret gs.

TABLE 4: Computation complexity

Ref.	Sharing	Verfication		Reconstruction	
	Exp.	Exp.	Pair	Exp.	Pair
$[15]_{DBS}$	$2n$	n	$2n$	$t+1$	$2t+1$
$[15]_{DDH}$	$4n$	$5n$	–	$5t+3$	0
$[16]_{ALBATROSS}$	$2n$	$2n$	–	$6t$	–
$[17]_{HEPVSS}$	$7n$	$4n$	–	$3t$	–
$[17]_{DHPVSS}$	$n(n-t+2)+2$	$n(n-t)+4$	–	$5t$	–
Ours	$2n$	$2n$	–	t	$2t$

TABLE 5: Communication complexity

Ref.	Sharing		Reconstruction	
	$\mathbb{G}$	$\mathbb{Z}_p$	$\mathbb{G}$	$\mathbb{Z}_p$
$[15]_{DBS}$	$2n$	0	t	0
$[15]_{DDH}$	$4n$	$n+1$	$3t$	$t+1$
$[16]_{ALBATROSS}$	$2n$	$n+1$	$5t$	$4t$
$[17]_{HEPVSS}$	$3n$	$2n$	t	2
$[17]_{DHPVSS}$	$n+2$	1	$3t$	t
Ours	$2n$	$n+2$	t	0

By Table 4 and Table 5, we compare the complexity between our proposed PVSS scheme and previous schemes [15], [16], [17], where n is the number of shareholders and t is the threshold value. Appendix D presents the detailed complexity analysis of previous PVSS schemes.

7. Implementation and Evaluation

We implement the proposed PVSS scheme with Charm-Crypto library, which is a framework for constructing cryptographic schemes. The Charm-Crypto framework is written in Python language, however it has plausible performance due to reliance of the GMP library [52] and the PBC library [53]. We choose an asymmetric curve *MNT159* to implement the proposed PVSS scheme. We implement a fully-connected peer-to-peer (p2p) network based on TCP socket programming to enable pair-wise communication. Both the consumer and the producer processes share the same P2P network interface, thereby competing for network resources. Our experiments are executed on 128 AWS cloud servers t4g.medium scattered in 8 regions, namely, Canada, Ireland, Ohio, Paris, SaoPaulo, Seoul, Singapore and Sydney. Each of the servers is with 2 vCPUs and 4 GB RAM (similar to SPURT [34]) and runs Linux ubuntu-bionic-18.04 with Python 3.6.9.

TABLE 6: Cryptographic cost

$\text{Exp}_{\mathbb{G}_0}$	$\text{Exp}_{\mathbb{G}_1}$	Pair	$\lvert\mathbb{G}_0\rvert/\lvert\mathbb{G}_1\rvert$	$\lvert\mathbb{Z}_p\rvert$
0.46ms	4ms	3.6ms	100B/304B	48B

We begin by evaluating the cryptographic operation costs of the curve *MNT159*, as shown in Table 6. Each element on $\mathbb{G}_0, \mathbb{G}_1, \mathbb{Z}_p$ element costs 100 Bytes, 304 Bytes and 48 Bytes, respectively. It is evident that the bilinear pairing ($e : \mathbb{G}_0 \times \mathbb{G}_1 \to \mathbb{G}_T$) and $\text{Exp}_{\mathbb{G}_1}$ incur a significantly higher cost than $\text{Exp}_{\mathbb{G}_0}$.[2]

We then assess the performance of our proposed PVSS scheme in comparison with related PVSS schemes. Figures 8, 9, and 10 illustrate the computation costs for sharing, verification, and reconstruction phases, respectively. Since $\mathbb{G}_1$ (cf. $\text{Exp}_{\mathbb{G}_1}$ in Table 6) is used in SCRAPE_{DBS} [15], it results in the highest computation overhead in the distribution phase. The results also suggest that ALBATROSS [16], which builds upon SCRAPE_{DDH} [15], shifts the computational burden from the sharing and verification phases to the reconstruction phase. The superlinear costs in sharing (cf. Figure 8) and verification (cf. Figure 9) phases of DHPVSS arise from evaluating a random (n-t-1)-degree polynomial at each i, where $i \in [1, \cdots, n]$, leading to approximately $O(n^2)$ exponentiations. SCRAPE_{DBS} and our PVSS incur higher computational costs than HEPVSS, DHPVSS and SCRAPE_{DDH} in the reconstruction phase (see Figure 10), primarily because they use bilinear pairings instead of DLEQ for verifying decrypted PVSS shares.

Figures 11 and 12 compare the communication overhead during the sharing and reconstruction phases for the dealer and the recoverer as the number of parties n increases. Our PVSS scheme demonstrates competitive performance in terms of communication efficiency.

Next, we evaluate the performance of AsyRand under different configurations with $n = 16, 32, 64, 128$. The

2. In some programming libraries or on certain elliptic curves, such as $SS512$, the bilinear pairing is faster than $\text{Exp}_{\mathbb{G}_0}$.

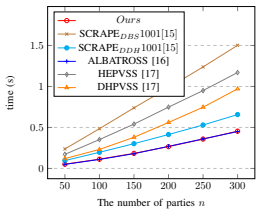


Figure 8: Comp. cost in the sharing

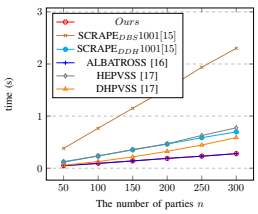


Figure 9: Comp. cost of in the verification

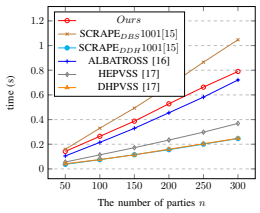


Figure 10: Comp. cost in the reconstruction

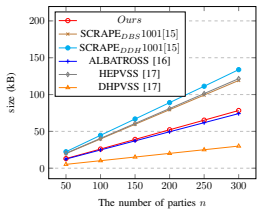


Figure 11: Comm. cost in the sharing

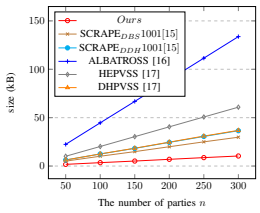


Figure 12: Comm. cost in the reconstruction

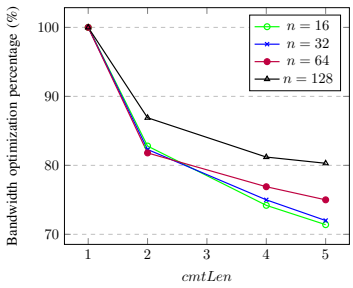


Figure 13: Bandwidth optimization with multiple PVSS commitments broadcast

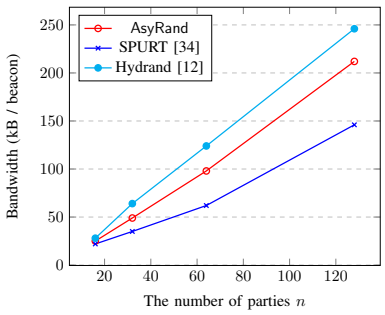


Figure 14: Bandwidth per beacon

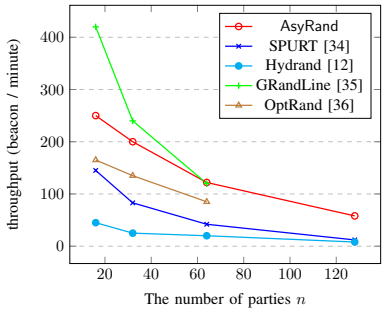


Figure 15: Average beacon throughput per minute

producer-consumer model decouples the production and consumption of PVSS commitments, making it convenient to identify performance bottlenecks. As expected, the producer process operates faster than the consumer process. Theorem 9 demonstrates that AsyRand beacon values are generated at the speed of the consumer process. To prevent the consumer process from lagging in computational resource allocation, we introduce parameters to regulate the producer's speed. These parameters include *queLen* (the maximum queue length) and $cmtLen$ (the number of PVSS commitments sent at a time by a party). Adjusting *queLen* effectively slows down the producer process, while modifying $cmtLen$ reduces bandwidth consumption, as multiple PVSS commitments can be transmitted via a single RBC.

By setting *queLen* to 3, the producer process is slowed with minimal impact on throughput. Adjusting $cmtLen$ to 2, 4, or 5 reduces bandwidth by approximately 17%-27% compared to $cmtLen = 1$. Figure 13 shows the results of bandwidth optimization.

Figure 14 illustrates the bandwidth usage (both sent and received data) per beacon output. For $n = 32$ with $queLen = 2$ and $cmtLen = 4$, the average bandwidth of a party is around 51kB per beacon. The bandwidth usage in AsyRand is higher than in SPURT [34] (35 kB per beacon), because two consensus algorithms are occurring independently in both the producer and consumer processes. However, this increased bandwidth does not hinder AsyRand's high throughput in an asynchronous setting.

As illustrated by Figure 15, AsyRand achieves significantly higher throughput compared to Hydrand [12], SPURT [34] and OptRand [36]. Specifically, with $n = 128$, AsyRand produces 58 beacons per minute, while SPURT and Hydrand produce only 12 and 8 beacons per minute, respectively. The throughput of GRandLine [35] exhibits significant degradation as n increases, likely due to its reliance on synchronous network assumptions. In synchronous network, network delay upper bound is usually hard-coded. Though GRandLine has a high throughput when $n < 64$, but it might become insecure if the predetermined time bound is exceeded.

8. Conclusion

In this paper, we introduce AsyRand, a distributed randomness beacon protocol designed for asynchronous network setting. AsyRand operates within a producer-consumer model. Each producer process generates PVSS commitments, which are then consumed by all consumer processes to continuously produce beacon values. To reach consensus among distributed parties, we integrate RBC and t-VABA protocols in producer and consumer, respectively. Additionally, AsyRand supports reconfiguration, enabling the removal of faulty parties and the addition of new parties without system restart. Our analysis establishes that AsyRand achieves key properties, including liveness, unpredictability, bias-resistance, public verifiability and responsiveness. Experimental results further validate the feasibility and effectiveness of AsyRand. As an independent contribution, we introduce an innovative PVSS scheme based on Sigma protocol and the Fiat-Shamir heuristic. In the future, we plan to explore verifiable batched secret sharing [44] to further reduce communication overhead and improve throughput. We will also explore new t-VABA protocols and the producer-consumer model in implementing asynchronous BFT consensus.

References

[1] Rabin, M. O. Transaction protection by beacons. *J. Comput. Syst. Sci.*, 1983, 27(2):256–267.

[2] Escudero, D., Tjuawinata, I., & Xing, C. On Information-Theoretic Secure Multiparty Computation with Local Repairability. In *PKC*, 2024, pp. 205-239.

[3] David, B., Magri, B., Matt, C., Nielsen, J. B., & Tschudi, D. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *CCS*, 2023, pp. 683-696.

[4] Hu, D., Wang, J., Liu, X., Li, Q., & Li, K. LMChain: An Efficient Load-Migratable Beacon-based Sharding Blockchain System. *IEEE TC*, 2024.

[5] Hou, R., Yu, H., & Saxena, P. Using throughput-centric byzantine broadcast to tolerate malicious majority in blockchains. In *S&P*, 2022, pp. 1263-1280.

[6] Kiayias, A., Russell, A., David, B., & Oliynykov, R. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO*, 2017, pp. 357-388.

[7] Van Den Hooff, J., Lazar, D., Zaharia, M., & Zeldovich, N. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *POSP*, 2015, pp. 137-152.

[8] Abraham, I., Chan, T. H., Dolev, D., Nayak, K., Pass, R., Ren, L., & Shi, E. Communication complexity of byzantine agreement, revisited. In *PODC*, 2019, pp. 317-326.

[9] David, B., Gaži, P., Kiayias, A., & Russell, A. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT*, 2018, pp. 66-98.

[10] Choi, K., Manoj, A., & Bonneau, J. SoK: Distributed randomness beacons. In *S&P*, 2023, pp. 75-92.

[11] Bonneau, J., Clark, J., & Goldfeder, S. On bitcoin as a public randomness source. http://eprint.iacr.org/2015/1015, 2015.

[12] Schindler, P., Judmayer, A., Stifter, N., & Weippl, E. HydRand: Efficient Continuous Distributed Randomness. In *S&P*, 2020, pp. 73-89.

[13] Syta, E., Jovanovic, P., Kogias, E. K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M. J., & Ford, B. Scalable Bias-Resistant Distributed Randomness. In *S&P*, 2017, pp. 444–460.

[14] Bünz, B., Goldfeder, S., & Bonneau, J. Proofs-of-delay and randomness beacons in Ethereum. In *S&B*, 2017.

[15] Cascudo, I., & David, B. SCRAPE: Scalable Randomness Attested by Public Entities. In *ACNS*, 2017, pp. 537-556.

[16] Cascudo, I., & David, B. ALBATROSS: Publicly Attestable Batched Randomness Based on Secret Sharing. In *ASIACRYPT*, 2020, pp. 311-341.

[17] Cascudo, I., David, B., Garms, L., & Konring, A. YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model. In *ASIACRYPT*, 2022, pp. 651-680.

[18] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP*, 2017, pp. 51-68.

[19] Hanke, T., Movahedi, M., & Williams, D. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[20] Cascudo, I., David, B., Shlomovits, O., & Varlakov, D. Mt. Random: Multi-tiered randomness beacons. In *ACNS*, 2023, pp. 645-674.

[21] Das, S., Yurek, T., Xiang, Z., Miller, A., Kokoris-Kogias, L., & Ren, L. Practical Asynchronous Distributed Key Generation. In *S&P*, 2022, pp. 2518-2534.

[22] Boneh, D., Bonneau, J., Bünz, B., & Fisch, B. Verifiable delay functions. In *Crypto*, 2018, pp. 757-788.

[23] Schoenmakers, B. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *CRYPTO*, 1999, pp. 148-164.

[24] Zhang, L., Qiu, F., Hao, F., & Kan, H. 1-round distributed key generation with efficient reconstruction using decentralized CP-ABE. *IEEE TIFS*, 2022, 17: 894-907.

[25] Abraham, I., Bacho, R., Loss, J. and Stern, G. Nearly Quadratic Asynchronous Distributed Key Generation. Cryptology ePrint Archive. 2025.

[26] Castro, M., & Liskov, B. Practical byzantine fault tolerance. In *OSDI*, 1999.

[27] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., & Abraham, I. HotStuff: BFT consensus with linearity and responsiveness. In *PODC*, 2019, pp. 347-356.

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

[29] Damgård, I. On Σ-Protocols, https://www.cs.au.dk/~ivan/Sigma.pdf, Accessed: 2025-03-06

[30] Fiat, A., & Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986, pp. 186–194.

[31] Shamir, A. How to share a secret. *Comm. of the ACM*, 1979, 22(11): 612–613.

[32] Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987, pp. 427-438.

[33] Bhat, A., Shrestha, N., Luo, Z., Kate, A., & Nayak, K. Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *CCS*, 2021, pp. 3502-3524.

[34] Das, S., Krishnan, V., Isaac, I. M., & Ren, L. Spurt: Scalable distributed randomness beacon with transparent setup. In *S&P*, 2022, pp. 2502-2517.

[35] Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S. and Papachristoudis, D. GRandLine: adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity. In *CCS*, 2024, pp. 941-955.

[36] Bhat, A., Shrestha, N., Kate, A., & Nayak, K. OptRand: Optimistically Responsive Reconfigurable Distributed Randomness. In *NDSS*, 2023.

[37] Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., & Weippl, E. Randrunner: Distributed randomness from trapdoor VDFs with strong uniqueness. In *NDSS*, 2021.

[38] Heidarvand, S., & Villar, J. L. Public verifiability from pairings in secret sharing schemes. In *SAC*, 2009, pp. 294-308.

[39] Chaum, D., & Pedersen, T. P. Wallet databases with observers. In *CRYPTO*, 1992, pp. 89-105.

[40] McEliece, R. J., & Sarwate, D. V. On sharing secrets and Reed-Solomon codes. *Comm. of the ACM*, 1981, 24(9), pp. 583-584.

[41] Stadler, M. Publicly verifiable secret sharing. In *Eurocrypt*, 1996, pp. 190-199.

[42] Cascudo, I. and David, B. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *Eurocrypt*, 2024, pp. 216-248.

[43] Meng, X., Sui, X., Yang, Z., Rong, K., Xu, W., Chen, S., Yan, Y. and Duan, S. Rondo: Scalable and Reconfiguration-Friendly Randomness Beacon. In *NDSS*, 2024.

[44] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S. and Stern, G., 2022. Bingo: Adaptively Secure Packed Asynchronous Verifiable Secret Sharing and Asynchronous Distributed Key Generation. In *CRYPTO*, 2023.

[45] Alhaddad, N., Das, S., Duan, S., Ren, L., Varia, M., Xiang, Z., and Zhang, H. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *PODC*, 2022, pp. 399-417.

[46] Guo, B., Lu, Z., Tang, Q., Xu, J., and Zhang, Z. Dumbo: Faster asynchronous bft protocols. In *CCS*, 2020, pp. 803-818.

[47] Abraham, I., Malkhi, D., and Spiegelman, A. Asymptotically optimal validated asynchronous byzantine agreement. In *PODC*, 2019, pp. 337-346.

[48] Yung, M. The "Mobile Adversary" Paradigm in Distributed Computation and Systems. In *PODC*, 2015, pp. 171-172.

[49] Fischer, M.J., Lynch, N.A. and Paterson, M.S. Impossibility of distributed consensus with one faulty process. *JACM*, 1985, 32(2):374–382.

[50] Mostéfaoui, A., Moumen, H. and Raynal, M. Signature-free asynchronous Byzantine consensus with $t < n/3$ and $O(n^2)$ messages. In *PODC*, 2014, pp. 2-9.

[51] Lu, Y., Lu, Z., Tang, Q. and Wang, G. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *PODC*, 2020, pp. 129-138.

[52] GMP library, https://gmplib.org, Accessed: 2024-07-05.

[53] PBC library, https://crypto.stanford.edu/pbc/, Accessed: 2024-07-05.

Appendix

1. Sigma Protocol and NIZK Proof

In a Sigma protocol [29], a prover ($\mathcal{P}$) demonstrates the validity of a statement x such that a verifier ($\mathcal{V}$) learns nothing about the witness w, where $(x, w) \in R$ represents a relation. A Sigma protocol follows a three-move interaction pattern between the prover and the verifier:

1) *Commitment* (a): $\mathcal{P}$ selects a random value r and computes a commitment a. $\mathcal{P}$ sends the a to $\mathcal{V}$.
2) *Challenge* (e): $\mathcal{V}$ selects a random challenge e from a challenge space and sends it to $\mathcal{P}$.
3) *Response* (z): $\mathcal{P}$ computes a response z using r, e, and the witness w. $\mathcal{P}$ sends the response z to $\mathcal{V}$. Then, $\mathcal{V}$ checks a verification equation involving a, e, and z.

In practice, NIZK proofs can be derived from the Sigma protocol by leveraging Fiat-Shamir heuristic [30], where the challenge value e is calculated by $\mathcal{P}$ with random oracle. Sigma protocols have the following properties:

- ***Completeness.*** If $\mathcal{P}$ knows the witness, $\mathcal{P}$ can always prove it.
- ***Soundness.*** If $\mathcal{P}$ does not know the witness, $\mathcal{P}$ cannot convince the verifier that it does.
- ***Zero-Knowledge.*** Zero-knowledge ensures that the verifier learns nothing beyond the validity of the statement.

2. Shamir secret sharing

Shamir's Secret Sharing [31] allows a dealer to distribute a secret among a group of parties. The underlying principle of Shamir's Secret Sharing is based on polynomial interpolation. Particularly, the dealer chooses a random (t-1)-degree polynomial $p(x) = s + \sum_{j=1}^{t-1} a_i x^j$, where s is the dealer's secret. Then, the shares are calculated as $s_i = p(i)$, for $i \in [1, \cdots, n]$. Each share s_i is sent to the corresponding shareholder P_i in a secret channel. Anyone who collects t shares can apply Lagrange interpolation to recover the secret, i.e., $s \leftarrow \mathsf{intpl}(\{(j, \tilde{p}(j))\}_{j\in[i_1,\cdots,i_t]})$. Nevertheless, if with less than t shares, no information about the secret is revealed.

3. Security Analysis of the Proposed PVSS

By Theorem 10, Theorem 11 and Theorem 12, we prove the required security properties, defined in Section 2.1, of the proposed PVSS scheme.

Theorem 10 (Correctness). Given at least t decrypted shares $\{D_i\}_{i\in T}$, the secret gs can be successfully recovered.

$Proof.$ The Lagrange interpolation ensures that $\sum_{i\in T} \mu_i \cdot p(i)$ interpolates to the value $p(0)$, where $\{\mu_i = \prod_{j\in T, j\neq i} \frac{j}{j-i}\}$ are the Lagrange coefficients. Consequently, the PVSS secret gs can be reconstructed as $g^s = g^{p(0)}$ by evaluating $\prod_{i\in T} D_i^{\mu_i}$ where $D_i = g^{p(i)}$. Therefore, the correctness of the PVSS scheme is guaranteed.

Lemma 7 (Completeness of Sigma protocol). If the dealer knows the secret s, he can prove it.

$Proof.$ The dealer, acting as the prover, generates (C, π_s) using $\mathsf{PVSS.Share}$ algorithm. Then the equations in the $\mathsf{PVSS.Verify}$ algorithm can be proved to be true as follows.

$$\begin{aligned} C_i' = \mathsf{pk}_i^{p'(i)} &= \mathsf{pk}_i^{\tilde{p}(i)} \mathsf{pk}_i^{c\cdot p(i)} = \mathsf{pk}_i^{\tilde{p}(i)} \cdot C_i^c,\ \forall i \in [1, \cdots, n] \\ \tilde{s} = \tilde{p}(0) &= \mathsf{intpl}(\{(i, p'(i) - c \cdot p(i))\}) \\ &= \mathsf{intpl}(\{(i, p'(i))\}) - \mathsf{intpl}(\{(i, c \cdot p(i))\}) \\ &= \mathsf{intpl}(\{(i, \tilde{p}(i))\}) \end{aligned}$$

Therefore, if the dealer knows the secret s, he can prove it in (C, π) with probability 1.

Lemma 8 (Soundness of Sigma Protocol). If the dealer does not know s, he cannot cheat the verifier successfully.

$Proof.$ It is widely adopted to prove Sigma Protocol ***Soundness*** by extracting the secret when two accepting proofs with the same commitment and different challenges are given [29]. Denote the two accepting proofs $\pi_1 = (C', c_1, (\tilde{s_1}, \{\tilde{p_1}(i)\}))$ and $\pi_2 = (C', c_2, (\tilde{s_2}, \{\tilde{p_2}(i)\}))$ for the statement C, where $c_1 \neq c_2$. Hence, we have:

$$\begin{cases} C_i' = \mathsf{pk}_i^{\tilde{p_1}(i)} \cdot C_i^{c_1} = \mathsf{pk}_i^{\tilde{p_1}(i) + c_1 \cdot p(i)} \\ C_i' = \mathsf{pk}_i^{\tilde{p_2}(i)} \cdot C_i^{c_2} = \mathsf{pk}_i^{\tilde{p_2}(i) + c_2 \cdot p(i)} \end{cases}, \forall i \in [1, \cdots, n]$$

By dividing the two equations, we get:

$$1_{\mathbb{G}} = \mathsf{pk}_i^{\tilde{p_2}(i) - \tilde{p_1}(i) + (c_2 - c1)\cdot p(i)}$$

Then $p(i)$ can be calculated as:

$$p(i) = \frac{\tilde{p_2}(i) - \tilde{p_1}(i)}{c_1 - c_2}$$

Further, $s \leftarrow \mathsf{intpl}(\{(i, p(i))\}_{i\in[1,\cdots,n]})$ and the secret s can be extracted with (C, π_1, π_2).

Theorem 11 (Public Verifiability). The messages from the dealer and the shares sent by shareholders are publicly verifiable.

Proof. By Lemma 7 and Lemma 8, the Sigma protocol used in the proposed PVSS provides a proof of knowledge for the dealer, who acts as the prover. Specifically, the output (C, π) of the PVSS.Share algorithm represents a non-interactive proof of knowledge for the secret s or gs. Moreover, the proof π can be verified publicly. Then we consider the public verifiability of shareholders in the reconstruction phase. Since C_i (in C) is already publicly verified and pk_i is publicly known, anyone can determine whether D_i from shareholder P_i is valid or not by $e(D_i, \mathsf{pk}_i) \stackrel{?}{=} e(g, C_i)$, as shown in the PVSS.Recon algorithm.

Lemma 9 (Zero Knowledge of Sigma Protocol). The output of PVSS.Share reveals no information about a shareholder P_i's $g^{p(i)}$, the dealer's secret gs.

Proof. We introduce a simulator $\mathcal{S}$, taking input a valid statement $C \in \mathbb{G}$ and a challenge $c \in \mathbb{Z}_p$. We prove that $\mathcal{S}$ can produce an accepting proof $(C', c, (\tilde{s}, \{\tilde{p}(i)\}))$ for C. Moreover, the proof should have the same distribution as a transcript generated by a real-world prover and verifier.

In the PVSS.Share algorithm of the real world, an honest prover can always output a transcript $(C', c, (\tilde{s}, \{\tilde{p}(i)\}))$, which is randomly distributed. Then, we argue the output of the simulator $\mathcal{S}$. For $\forall i \in [1 \cdots, n]$, it firstly calculates :

$$p_t(i) \xleftarrow{R} \mathbb{Z}_p, C_{it} \leftarrow \mathsf{pk}_i^{p_t(i)} \cdot C_i^c$$

and outputs a tuple $(C_{it}, c, p_t(i))$. Notice that the tuple always represents an accepting proof, as required. Further, since c and $p_t(i)$ are randomly distributed in $\mathbb{Z}_p$, it follows that C_{it} is also randomly distributed in $\mathbb{G}$. Summarily, the simulator $\mathcal{S}$ can always output a transcript that is indistinguishable from the output of real-world prover and verifier, meaning that nothing about $p(i)$ is leaked. Hence, the adversary cannot obtain the P_i's $g^{p(i)}$ without $p(i)$. Besides, it is also infeasible to defer $g^{p(i)}$ using $C_i' = \mathsf{pk}^{p(i)} = (g^{p(i)})^{\mathsf{sk}_i}$, due to discrete logarithm problem.

Definition 2 (IND1-Secrecy Game). A PVSS scheme achieves ***IND1-Secrecy*** if for any polynomial time adversary $\mathcal{A}$ corrupting at most $t-1$ parties, $\mathcal{A}$ has negligible advantage in the following game.

1) A challenger $\mathcal{C}$ runs the PVSS.Setup algorithm and sends $(g, \mathsf{pk}_i, \mathsf{sk}_i)$ to each uncorrupted shareholder P_i and all public information to $\mathcal{A}$.
2) $\mathcal{A}$ creates secret keys for the $t-1$ corrupted parties and sends the corresponding public keys $\{\mathsf{pk}_i\}$ to the challenger $\mathcal{C}$.
3) $\mathcal{C}$ selects two random values $x_0, x_1 \in \mathbb{G}$ and randomly chooses $b \leftarrow \{0, 1\}$. It then runs the PVSS.Share algorithm with secret x_0 and sends all the output to $\mathcal{A}$, along with x_b.
4) $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$.

$\mathcal{A}$'s advantage over the game is defined as $|\Pr[b = b']\text{-}1/2|$.

Theorem 12 (IND1-Secrecy). The proposed PVSS scheme achieves ***IND1-Secrecy***, i.e., for any probabilistic polynomial time adversary $\mathcal{A}$, corrupting fewer than t shareholders, has a negligible advantage in obtaining information about gs.

Proof. By Lemma 9, the Sigma protocol used in the proposed PVSS provides zero knowledge about each $g^{p(i)}$ for $i \in [1, \cdots, n]$ in the PVSS.Share phase. Then, we consider the situation where $\mathcal{A}$ corrupts $t-1$ shareholders. We prove the **IND1-Secrecy** property by analyzing the security game defined in Definition 2 based on discrete decisional Diffie-Hellman (DDH) assumption. We argue that, if $\mathcal{A}$ can break the **IND1-Secrecy** property of our PVSS, then there exists an adversary $\mathcal{A}_{DDH}$ which can use $\mathcal{A}$ to break DDH assumption. Without lose of generality, denote the first $t-1$ shareholders, i.e., $[P_1, \cdots, P_{t-1}]$, as the corrupted parties.

Let $(g, g^\alpha, g^\beta, g^\gamma)$ be an instance of the DDH problem. Then, $\mathcal{A}_{DDH}$ using $\mathcal{A}$ can sumulate an security game as follows:

1) The challenger $\mathcal{C}$ sets $h = g^\alpha$. Then, $\mathcal{C}$ runs the PVSS.Setup algorithm and sends $(g, \mathsf{pk}_i, \mathsf{sk}_i)$ to each uncorrupted shareholder $P_i \in [t, \cdots, n]$. For $t \le i \le n$, $\mathcal{A}_{DDH}$ samples $r_i \xleftarrow{R} \mathbb{Z}_p$ and sends $\mathsf{pk}_i = g^{r_i}$ to $\mathcal{A}$.
2) For $1 \le i \le t-1$, $\mathcal{A}$ chooses uniformly random values $\mathsf{sk}_i \xleftarrow{R} \mathbb{Z}_p$ and sets $\mathsf{pk}_i = h^{\mathsf{sk}_i}$ and sends these to the challenger.
3) For $1 \le i \le t-1$, $\mathcal{C}$ chooses uniformly random values $s_i \leftarrow \mathbb{Z}_p$ and sets $C_i = \mathsf{pk}_i^{s_i}$.
   For $t \le i \le n$, $\mathcal{C}$ generates $S_i = g^{p(i)}$ where $p(x)$ is the (t-1)-degree polynomial determined by $p(0) = \beta$ and $p(j) = s_j$ for $1 \le j \le t-1$. Note that $\mathcal{A}_{DDH}$ knows g^β (but does not know β) and g^{s_j} for $1 \le j \le t-1$. So $\mathcal{A}_{DDH}$ can use Lagrange interpolation to computes $S_i = g^{s_i} = g^{p(i)}$ for $t \le i \le n$ and it also generates shares $C_i = (S_i)^{r_i} = \mathsf{pk}_i^{s_i}$. Denote $C = \{C_i\}_{i \in [1, \cdots, n]}$. The challenger calculates the NIZK proofs π for C, as the dealer does. Finally, $\mathcal{C}$ sends (C, π) and g^γ to $\mathcal{A}$, where g^γ plays the role of x_b in the game.
4) Output: $\mathcal{A}$ makes a guess of b'.
   If $b' = 0$, $\mathcal{A}_{DDH}$ guess that $g^\gamma = \alpha \cdot \beta$, if $b' = 1$, $\mathcal{A}_{DDH}$ guess that γ is a random element. The information that $\mathcal{A}$ receives in setp 3) is distributed exactly like a sharing of the value $h^\beta = g^{\alpha\beta}$. If g^γ sent to $\mathcal{A}$ is the secret shared by the proposed PVSS, $\gamma = \alpha \cdot \beta$. So the advantage of $\mathcal{A}_{DDH}$ is the same as the advantage of $\mathcal{A}$.

4. Complexity of Related PVSS

In the following, we analyze the computation and communication complexity of some of the recent PVSS schemes [16], [17] in detail. To better elaborate the complexity, we first analyze the complexity of DLEQ algorithms, which is widely adopted in aforementioned PVSS schemes [15], [16], [17], [42]. The prover costs 2 exponentiations to generate commitments, outputting two elements in $\mathbb{G}$ and one element in $\mathbb{Z}$. The verification algorithm requires 4 exponentiations.

SCRAPE [15] has analyzed its complexities and we omit it here.

1) *ALBATROSS* [16]

Computation Complexity: In the sharing phase, the dealer costs one exponentiation to compute the secret S and n exponentiations to encrypt the shamir shares. In addition, it also takes n exponentiations to produce the low degree exponent interpolation (LDEI) proof, which essentially executes a standard Sigma protocol for each share. Thus, the total cost in the sharing phase is $2n$ exponentiations. In the verification phase, the verifier costs $2n$ exponentiations to check the LDEI proof. In the reconstruction phase, the cost includes 1 exponentiation to verify the local LDEI proof and 4 exponentiations to verify the DLEQ proof for each party providing a decrypted share. Lastly, reconstructing the secret S requires t exponentiations. Hence, the reconstruction phase takes $6t$ exponentiations.

Communication Complexity: In the sharing pahse, the dealer publishes the encrypted shares $(\hat{\sigma}_1, ..., \hat{\sigma}_n)$ on the public ledger along with the proof LDEI. The encrypted shares contains n elements in $\mathbb{G}$, and the proof LDEI contains n elements in $\mathbb{G}$ and $n+1$ elements in $\mathbb{Z}$. In the reconstruction phase, the secret S can be reconstructed only if at least t shares are published in the ledger. So, t shares contains t elements in $\mathbb{G}$. In addition, each party must publish DLEQ proof and the encrypted share to show that the decrypted share corresponds to $\hat{\sigma}_i$, in total t DLEQ proofs and t encrypted shares. Hence, reconstruction phase requires $4t$ elements in $\mathbb{G}$ and t elements in $\mathbb{Z}$.

2) *HEPVSS* [17]

Computation Complexity: In the sharing phase, the dealer costs n exponentiations to generate the shares $\{A_i : i \in [n]\}$ and $2n$ exponentiations to encrypt these shares using ElGamal encryption, resulting in $\{C_i : i \in [n]\}$. Besides, generating the corresponding NIZK proof Pf_{sh} necessitates $4n$ exponentiations. Therefore, the total computation cost for the sharing phase is $7n$ exponentiations. In the verification phase, the verifier takes $4n$ exponentiations to verify the NIZK proof Pf_{sh} generated in the sharing phase. In the reconstruction phase, it takes $2t$ exponentiations to verify the proof Pf_{Dec}, which is generated by all shareholders to guarantee the correctness of the decrypted shares. Finally, t exponentiations are cost to reconstruct the secret S. Hence, the reconstruction phase takes $3t$ exponentiations, in total.

Communication Complexity: In the sharing phase, the dealer publishes the encrypted shares $\{C_i : i \in [n]\}$ and the corresponding NIZK proof Pf_{sh}. The set $\{C_i : i \in [n]\}$ contains $2n$ elements in $\mathbb{G}$ and the proof Pf_{sh} contains n elements in $\mathbb{G}$ and $2n$ elements in $\mathbb{Z}$. the communication complexity for the sharing phase is $3n$ elements in $\mathbb{G}$ and $2n$ elements in $\mathbb{Z}$. In the reconstruction phase, the recoverer receives t shares and the corresponding NIZK proof Pf_{Dec} to reconstruct the secret S. Each share contains one elements in $\mathbb{G}$, while Pf_{Dec} contains $2t$ elements in $\mathbb{Z}$ in total. Thus, the communication complexity for the reconstruction phase is t elements in $\mathbb{G}$ and $2t$ elements in $\mathbb{Z}$.

3) *DHPVSS* [17]

Computation Complexity: In the sharing phase, the dealer costs n exponentiations to generate the shares $\{A_i : i \in [n]\}$ and an additional n exponentiations to generate the encrypted shares $\{C_i : i \in [n]\}$. Then, the dealer leverages DLEQ to generate the single proof $DLEQ(sk_D; G, pk_D, U, V)$. The computation of U and V requires n exponentiations each, and involves evaluating a random $(n-t-1)$-degree polynomial at i, for $i \in [1, \ldots, n]$, which takes $n(n-t-2)$ exponentiations. Therefore, the total cost for computing U and V is $n(n-t-2)+2n$ exponentiations. Hence, the sharing phase takes $n(n-t+2)+2$ exponentiations. In the verification phase, the verifier needs to recompute the U and V and verify the DLEQ proof Pf_{sh}. Hence, the verification phase incurs a cost of $n(n-t)+4$ exponentiations. In the reconstruction phase, verifying the t DLEQ proofs costs $4t$ exponentiations, and reconstructing the secret S requires an additional t exponentiations. Hence, the total computation cost for the reconstruction phase is $5t$ exponentiations.

Communication Complexity: In the sharing phase, the dealer publishes $(\{C_i : i \in [n]\}, Pf_{sh})$, where $\{C_i : i \in [n]\}$ contains n elements in $\mathbb{G}$, and Pf_{sh} contains 2 elements in $\mathbb{G}$ and one element in $\mathbb{Z}$. Hence, the communication complexity of the sharing phase is $n+2$ elements in $\mathbb{G}$ and one element in $\mathbb{Z}$. In the reconstruction phase, the recoverer receives t shares and t DLEQ proofs. Each share is an element in $\mathbb{G}$. Therefore, the communication complexity of the reconstruction phase is $3t$ elements in $\mathbb{G}$ and t elements in $\mathbb{Z}$.