$AsyRand$: fast asynchronous distributed randomness beacon with reconfiguration

Liang Zhang, Tao Liu, Zhanrong Ou, Haibin Kan* *Member, IEEE*, and Jiheng Zhang* . . .

***Abstract*—Distributed randomness beacon protocols, which generate publicly verifiable randomness at regular intervals, are crucial for a wide range of applications. The publicly verifiable secret sharing (PVSS) scheme is a promising cryptographic primitive for implementing beacon protocols, such as Hydrand (S&P '20) and SPURT (S&P '22). However, two key challenges for practical deployment remain unresolved: asynchrony and reconfiguration. In this paper, we introduce the $AsyRand$ beacon protocol to address these challenges. In brief, $AsyRand$ leverages Bracha Reliable Broadcast (BRB) or BRB-like protocols for message dissemination and incorporates a producer-consumer model to decouple the production and consumption of PVSS commitments. In the producer-consumer model, PVSS commitments are produced and consumed using a queue data structure. Specifically, the producer process is responsible for generating new PVSS commitments and reaching consensus on them within the queue, while the consumer process continuously consumes the commitments to recover PVSS secrets and generate new beacon values. This separation allows the producer and consumer processes to operate simultaneously and asynchronously, without the need for a global clock. Moreover, the producer-consumer model enables each party to detect potential faults in other parties by monitoring the queue length. If necessary, parties in $AsyRand$ can initiate a removal process for faulty parties. BRB is also employed to facilitate the addition of new parties without requiring a system restart. In summary, $AsyRand$ supports reconfiguration, enhancing both the protocol's usability and reliability. Additionally, we propose a novel PVSS scheme based on the Σ protocol, which is of independent interest. Regarding complexity, $AsyRand$ achieves state-of-the-art performance with $O(n^2)$ communication complexity, $O(n)$ computation complexity, and $O(n)$ verification complexity.**

***Index Terms*—distributed randomness beacon, publicly verifiable secret sharing, producer-consumer, reliable broadcast**

I. Introduction

Distributed randomness beacon protocols are designed to generate sequences of trustworthy and verifiable random values at regular intervals [1]. Beacons have a broad spectrum of applications. Notable examples include secure multiparty computation [2], consensus protocols [6], [19], anonymous communication [8], [7], blockchain sharding [3], [4] and byzantine agreement protocol [5] in distributed systems.

A beacon protocol must ensure that random values are both unpredictable and unbiased prior to generation, while also being publicly verifiable once generated. The key properties of a beacon [14], [13] are summarized as follows:

- *Liveness/Availability:* The beacon must be capable of producing randomness outputs at regular intervals, even in the presence of adversarial activity or failures.
- *Bias resistance:* Each randomness must be unbiased and uniformly distributed. This ensures that no adversary can influence the random value in a predictable manner.
- *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.
- *Guaranteed output delivery:* Malicious parties cannot prevent the correct parties from generating beacon values.

A. Related works

Randomness obtained from centralized protocols, such as NIST randomness beacon[10], might be manipulated. Compared with centralized approaches, distributed methods enhance security of randomnesses by eliminating the single points of failure, but they introduce additional complexities in coordination and consensus among distributed parties [11].

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 [12]. Other cryptographic primitives are also employed as the underlying tool of beacon protocols, such as verifiable randomness function (VRF) [19], [9], verifiable delay function (VDF) [23], [37], threshold signatures [20], verifiable secret sharing (VSS) [32], [33], [42] and publicly verifiable secret sharing (PVSS) [16], [13], [34], [35]. Some beacon protocols [20], [21], also necessitate distributed key generation (DKG) [21], [25], [22], [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 [40], [24], [38], [16], [17] are advantageous as they do not require a private communication chan-

Liang Zhang and Jiheng Zhang is with Department of Industrial Engineering and Decision Analytics, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. scottzhang@ust.hk; jiheng@ust.hk;

Tao Liu and Zhanrong Ou are with the School of Cyberspace Security (School of Cryptology), Hainan University, Haikou 570228, China. jb8181632@gmail.com; ozr@hainanu.edu.cn;

Haibin Kan is with School of Computer Science, Fudan University, Shanghai 200433, China. hbkan@fudan.edu.cn

This work was supported the National Natural Science Foundation of China for Young Scientists (No. 62302129), the National Natural Science Foundation of China (No. 62272107), Hainan Province Key R&D plan project (No. ZDYF2024GXJS030) and the HK RGC General Research Fund (Nos. 16208120 and 16214121).

*Corresponding author

TABLE I: Comparison of PVSS-based beacon protocols

Protocol	Network	Live.	Comm.	Unpred.	Bias-rist.	Comp.	Veri.	Reconf.	Resp.
RandHerd [14]	asyn.	✗	$O(c^2 \text{log} n)$	✓	✗	$O(c^2 \text{log} n)$	$O(1)$	✗	✗
SCRAPE [16]	syn.	✓	$O(n^3)$	✓	✓	$O(n^2)$	$O(n^2)$	✗	✗
HydRand [13]	syn.	✓	$O(n^2)$	✓$^{\nearrow}$	✓	$O(n)$	$O(n)$	✗	✗
GRandPiper [33]	syn.	✓	$O(n^2)$	✓$^{\nearrow}$	✓	$O(n^2)$	$O(n^2)$	✓	✗
SPURT [34]	semi-syn.	✓	$O(n^2)$	✓	✓	$O(n)$	$O(n)$	✗	✓
OptRand [36]	syn.	✓	$O(n^2)$	✓	✓	$O(n)$	$O(n)$	✓	✓
GRandLine [35]	syn.	✓	$O(n^2)$	✓	✓	$O(n)$	$O(n)$	✗	✓
$AsyRand$	asyn.	✓	$O(n^2)$	✓$^{\nearrow}$	✓	$O(n)$	$O(n)$	✓	✓

$\nearrow$ means that 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. Additionally, the unpredictability might be broken slightly in $AsyRand$ in the removal or joining process, as analyzed in Section IV-D and Section IV-E.

nel and can identify faulty parties. RandHerd [14] divides parties into c-size subgroups to achieve scalability with communication and computation complexity $O(c^2 logn)$ 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 [16], 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 [13], 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 byzantine faulty parties, respectively. SPURT improves communication efficiency by aggregating PVSS commitments based on SCRAPE PVSS within a semi-synchronous network. 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. Table I provides a comparison of PVSS-based beacon protocols.

This paper addresses the challenge remaining in previous PVSS-based beacon research: achieving asynchrony while supporting reconfiguration.

B. Our approach in a nutshell

We take advantage of the Bracha reliable broadcast (BRB) [29] protocol to build our beacon protocol, where BRB is regarded as a primary primitive to build asynchronous consensus [28]. Inspired by the resource-management mechanisms of operating systems, we leverage the producer-consumer model to manage PVSS commitments for distributed parties.

Particularly, the producer process of each party continuously generates PVSS commitments and achieves consistency among all parties by utilizing BRB protocol. These PVSS commitments are managed in separate queues for each party. The consumer process defines the concept of epoch, where a leader is randomly elected and a random beacon is collectively produced. Specifically, the leader for the current epoch is randomly selected using the randomness beacon value from the previous epoch. The beacon value for the current epoch is then computed using the leader's earliest unconsumed PVSS secret as the random source. Thus, the primary task of the consumer process is to recover an unconsumed PVSS secret for the randomly chosen leader. To ensure consistency within the consumer process, we design a BRB-like protocol that differs from the traditional BRB protocol by operating without a predefined leader.

Refer to Appendix A for the detailed benefits of the producer-consumer model, which facilitates asynchrony and reconfiguration.

C. Contributions

We propose a novel beacon protocol, $AsyRand$, described in a distributed producer-consumer model. The producer process consists of an infinite execution of the BRB protocol, while the consumer process is designed with continuous instances of BRB-like protocol. Both processes achieve BFT consensus assuming an asynchronous network, which might be independent interest to BFT researchers.

To handle party-renewal without system restart, we design reconfiguration by incorporating a removal process and a joining process. Both processes are rigorously analyzed based on the properties of BRB or BRB-like protocol. To our knowledge, this is the first time to treat beacon reconfiguration in an asynchronous environment.

We design a new PVSS scheme maintaining state-of-the-art complexity, where the Σ-protocol [30] and Fiat-Shamir heuristic [31] are leveraged to achieve non-interactive zero knowledge (NIZK) proofs. It might be of independent interest.

The security requirements of the four processes are formally proved and properties of the beacon are thoroughly discussed. Comprehensive experiments are conducted to evaluate the throughput and bandwidth of proposed beacon protocol. With $n = 32$, the throughput reaches 200 beacons per minute, significantly outperforming SPURT [34] (84 beacons per minute, S&P'22) and Hydrand [13] (25 beacons per minute, S&P'20).

II. Preliminaries

A. Publicly verifiable secret sharing (PVSS)

A publicly verifiable secret sharing (PVSS) enables a dealer to share secrets among distributed shareholders in a publicly verifiable manner. Particularly, the dealer shares a secret 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., $C = \{C_i\}$ and π.
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) $g^s \leftarrow \mathsf{PVSS.Recon}(C, \{D_i\}_{i \in T})$ With a set of decrypted shares T ($|T| \geq t$), the secret value g^s can be reconstructed. Moreover, the decrypted shares $\{D_i\}_{i \in T}$ are also publicly verifiable.

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 from the dealer. Besides, the decrypted shares should also be publicly verifiable in the reconstruction phase.
- ***IND1-Secrecy*** IND1-Secrecy [38] guarantees that an adversary, given any $t-1$ secret keys $\{\mathsf{sk}_i\}$ and the public information, learns no information about the secret g^s.

B. Bracha reliable broadcast

Bracha reliable broadcast (BRB) [29] ensures that a message broadcast by an honest party is received by all non-faulty parties in a distributed system. Particularly, BRB 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. In our approach, we slightly modify the original consensus 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:

$Bracha\ Reliable\ Broadcast$ for $O(n)$-size v

Step 0. For the leader L
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 BRB protocol has the following security properties:

- ***Validity*** If the leader is non-faulty, then all non-faulty parties will output the leader's proposal v.[1]
- ***Agreement*** If some non-faulty party delivers v, then all non-faulty parties eventually delivers v.
- ***Termination*** If a non-faulty terminates with v, then all non-faulty parties will eventually terminates.

III. System model and goals

A. System model

The system operates under an authenticated asynchronous model with a network of pairwise connected public channels for broadcasting. Parties only accept and act upon a message only if it is correctly signed. Messages can be arbitrarily delayed in periods of asynchrony. The current number of parties ($\mathcal{P}$) in the system is denoted by variable n, i.e., $|\mathcal{P}| = n$. f represents the maximum number of faulty (also called malicious or byzantine) parties that may cause byzantine failures or disobey our protocol. A party is considered honest or non-faulty if it abides by the protocol; otherwise, it is considered to be faulty. We only consider one party joining or removal at a time in the reconfiguration. Actually, simultaneous reconfiguration degrades into sequential, one-by-one reconfiguration. Specifically, simultaneous joining violates the BRB **agreement** property, preventing any node from joining; simultaneous removal can be viewed as sequential removal with multiple instances of the removal process. The condition $f < n/3$ is assumed to hold, regardless of the number of parties in the system.

$AsyRand$ is introduced in a producer-consumer model, in which each party continuously produces new PVSS commitments and honest ones collectively consume the leader's earliest PVSS commitment. In the context of PVSS schemes, we set the threshold value $t = f+1$. An epoch is defined as the period during which a PVSS commitment is decrypted to produce a beacon value. The initial beacon value and the leader are assumed to be obtained in a decentralized way, which can be obtained with VDF [37] or a nonce from Bitcoin [12].

B. System goals and security threats

Our objective is to develop a beacon protocol, $AsyRand$, that supports reconfiguration in an asynchronous environment. To achieve this goal, we frame the protocol within a producer-consumer model for each party. In this model, both producer and consumer processes reach consensus using BRB or BRB-like protocol. Beacon values continuously output as the consumer process proceeds. The system goal is to achieve liveness, bias-resistance, unpredictability, public verifiability, guaranteed output delivery and responsiveness (beacon values output at the speed of real network).

In the system, malicious parties or adversaries can collude to violate the above properties by arbitrarily biasing the protocol.

[1]If leader is faulty in the BRB protocol, we prove that non-faulty parties will agree on $\bot$ by Lemma 1 in the producer process.

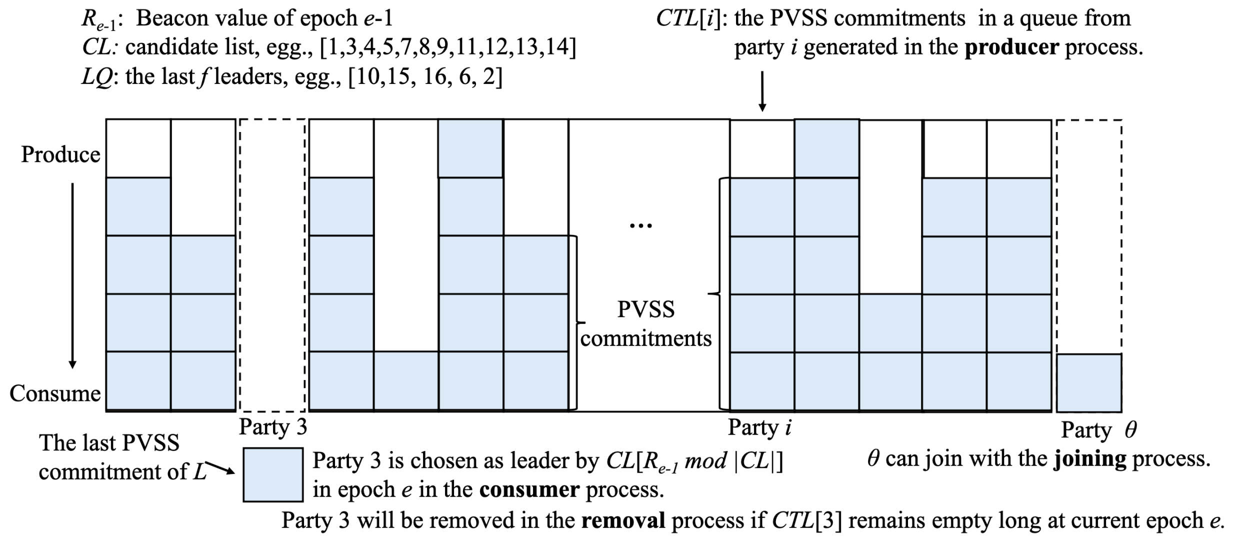
$R_{e\text{-}1}$: Beacon value of epoch e-1
CL: candidate list, egg., [1,3,4,5,7,8,9,11,12,13,14]
LQ: the last f leaders, egg., [10,15, 16, 6, 2]
$CTL[i]$: the PVSS commitments in a queue from party i generated in the **producer** process.
Produce
Consume
...
PVSS commitments
Party 3
Party i
Party θ
The last PVSS commitment of L
Party 3 is chosen as leader by $CL[R_{e\text{-}1} \bmod |CL|]$ in epoch e in the **consumer** process.
θ can join with the **joining** process.
Party 3 will be removed in the **removal** process if $CTL[3]$ remains empty long at current epoch e.

Fig. 1: Overview of the proposed beacon protocol $AsyRand$, depicting the four processes within a party

A summary of risks and potential attacks is introduced below: In the producer process, each malicious party has the following options: send invalid PVSS commitment (or nothing) to honest parties; send contradicting valid PVSS commitments to different honest parties; or delay the delivery of valid PVSS commitment. In the consumer process, a malicious party i may broadcast an invalid PVSS decryption key D_i (or nothing) when recovering a leader's secret value. Besides, the f malicious parties may collude to predict the random beacon values in advance.

IV. The $AsyRand$ beacon protocol

A. $AsyRand$ in producer-consumer model

Leveraging the PVSS scheme, we construct the $AsyRand$ protocol with a producer-consumer model for each party. The term "produce" refers to the generation of a PVSS commitment by a party, while "consume" refers to the reconstruction of the random value hidden within a PVSS commitment (i.e., the PVSS secret) by all parties. In the producer process, each party broadcasts new PVSS commitments, which are the outputs of the PVSS.Share algorithm, via continuously invoking BRB 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 BRB-like protocol. Further, the recovered random value is adopted to generate a fresh beacon value.

We define some global states/variables to enhance the clarity of our protocol. 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. Apparently, $CL = \mathcal{P} - LQ$. CTL is a dictionary and $CTL[i]$ is a queue recording the PVSS commitments of party i in chronological order. We use independent sequence numbers to guarantee chronology of PVSS commitments for each producer process, and it is omitted in the paper. Hence, all parties share the same global states, including $\{\mathsf{pk}_i\}, LQ, CL, R_{e-1}, e, L, CTL$. Figure 1 depicts all the processes and the global states of $AsyRand$ with concrete examples within a party. Table II presents 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 II: 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., $CL = \mathcal{P} - LQ$
CTL	dictionary of queues
$CTL[i]$	the queue recording unconsumed PVSS commitments from party i

The producer and the joining processes directly employ BRB protocols, while the consumer and the removal processes are based on BRB-like protocols. Additionally, the removal and the joining processes are one-shot. A summary of the processes designed in $AsyRand$ is provided in Section IV-F.

B. The producer process

The producer process directly employs the BRB protocol to allow each party to commit to secrets. Castro et al. [26] pointed out that BRB protocol does not resolve the problem when the leader is faulty. This situation is handled in our producer process. Each party commits to new random values by PVSS.Share which enable others to check honesty by PVSS.Verify. If the commitment is valid, the value can definitely be recovered or "consumed" in the consumer process.

In the producer process, each party i continuously invokes BRB protocol (cf. Section II-B) to commit new commitments as depicted by Figure 2. The main idea is for each party to commit to new random values using the PVSS scheme. Particularly, the proposal $v = (i, C, \pi)$ in **Step 0**, where (C, π) is output of the $\mathsf{PVSS.Share}$ algorithm. The public verifiability property of PVSS scheme enables anyone to check whether a BRB leader has honestly produced PVSS commitments by leveraging the $\mathsf{PVSS.Verify}$ algorithm in **Step 1**. Once a BRB instance is accomplished in **Step 3**, each party updates the global state by $CTL[v.i].put(v)$, where v is the PVSS commitment generated by the leader i. By leveraging the BRB properties (i.e., ***agreement***, ***validity***, ***termination***), we prove the BFT security requirements [26], [27] for the producer process in an asynchronous setting by Theorem 1 and Theorem 2.

```
Step 0. For party $i$, invokes $(C, \pi) \leftarrow \mathsf{PVSS.Share}$.
Denotes $v = (i, C, \pi)$ 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 $(\mathsf{echo}, hv)$, where $hv = Hash(v)$.

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

Step 3. For party $i$, waits until the receipt of
$2f + 1$ $(\mathsf{ready}, hv)$
  $CTL[v.i].put(v)$.
```

Fig. 2: The producer process at each party i

Lemma 1: Non-faulty parties reach agreement on "$\perp$" in the producer process if the leader is faulty.

Proof. If the PVSS dealer is faulty, non-faulty parties take no action to the queue $CTL[L]$. Thus, the queue length $|CTL[L]|$ is an indicator to judge whether the leader is honest or not. "$\perp$" is defined as the default agreement value, which is initialized by all non-faulty parties at the beginning of BRB protocol. If $|CTL[L]|$ remains unchanged, it means that the leader is faulty and agreement on "$\perp$" is achieved.

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

Proof. $CTL[i]$ is designed as a linear structure to record all PVSS commitments for each party i chronologically. If two values in the global state $CTL[i]$ are in wrong order, that means party i has successfully sent contradicting PVSS commitments in a proposal, which is impossible to achieve consensus due to the BRB **agreement** property.

*Theorem 1 (**Safety of the producer process**):* No matter whether the current leader L is faulty or not, the result of the producer process, $CTL[L]$, will always reach agreement for all honest parties.

Proof. The producer process consists of infinite BRB instances invoked by each party. Any two BRB instances led by two leaders are independent. Hence, we focus on the scenario where any party i invokes the BRB protocol. Namely, party i is the leader L of a BRB instance. No matter the leader L is faulty or not, all honest parties will output the same value, which may be the leader's proposal v or the initialized value "$\perp$" (cf. Lemma 1). If the consensus value is the former, the BRB ***termination*** property guarantees that the leader's proposal v will be eventually accepted and be appended to $CTL[L]$ for all honest parties. If the consensus value is the later, $CTL[L]$ will not be updated for all honest parties and the agreement of "$\perp$" does not impact the consensus of $CTL[L]$. Moreover, the elements in $CTL[L]$ are in right order by Lemma 2. Therefore, the global state $CTL[L]$ will be eventually under consensus for all honest parties.

*Theorem 2 (**Liveness of the producer process**):* No adversaries could prevent the producer process from appending valid PVSS commitment into $CTL[i]$ for honesty party i.

Proof. The producer process consists of infinite independent BRB 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., $CTL[i]$ and $CTL[j]$. We argue that the producer process keeps liveness by proving that $CTL[i]$ can be appended as long as i's PVSS commitments can be continuously provided. If the leader i is honest, i can initialize a BRB protocol once the previous BRB proposal has been agreed upon by all honest parties. The BRB ***termination*** property guarantees that all non-faulty parties will eventually reach a consensus on the leader's proposal. Therefore, i can initialize the next BRB proposal smoothly, allowing $CTL[i]$ to be appended with the new PVSS commitment. If the leader i is faulty, by the ***agreement*** and ***termination*** properties of the BRB protocol, two possible consensus outcomes may occur. If consensus is reached on some PVSS commitment, the result will be the same with the scenario where the leader is non-faulty. Therefore, the consensus value will be appended to $CTL[i]$. If consensus is reached on "$\perp$", $CTL[i]$ will keep unchanged, as stated in Theorem 1. The faulty leader will be removed from the system once it is deemed to have dropped out, which will be discussed in Section IV-D. The removal mechanism does not affect liveness for the remaining honest parties.

C. The consumer process

In the consumer process, all parties cooperate to recover L's earliest unconsumed PVSS commitment and reach consensus on a beacon value for each epoch in a BRB-like protocol. Figure 3 depicts the consumer process in epoch e for each party i. Different from the original BRB protocol, the protocol is led by all honest parties, rather than a leader.

In **Step 0**, each party obtains current leader by computing $L = CL[R_{e-1} \ mod \ |CL|]$, where R_{e-1} is the beacon value of previous epoch $e - 1$ and obtains the earliest PVSS commitment by $v \leftarrow CTL[L].get()$. Then, each party i generates the decryption key D_i for the commitment v. In **Step 1**, each party recovers the PVSS secret g^s with at least t valid decryption keys and constructs the beacon value R_e of current epoch e. The **Step 2** and **Step 3** are the same with the original BRB protocol.

The initialization message in **Step 0** is defined as (recon, D_i) where D_i is the decryption key obtained via $\mathsf{PVSS.PreRecon}(C_i, \mathsf{sk}_i)$. The echo message is defined as $(\mathsf{reconEcho}, e, R_e)$ in **Step 1**, where R_e is calculated by employing the recovered random value g^s of epoch e. Note that g^s was previously hidden in $v = (L, C, \pi)$ by the leader in the producer process. Thus, it is the unique PVSS secret and can be recovered with t valid keys. The ready message is defined as $(\mathsf{reconReady}, e, R_e)$ in **Step 2**.

```
Step 0. For party $i$, $L = CL[R_{e-1}\ mod\ |CL|]$ and $v \leftarrow CTL[L].get()$, where $v = (j, C, \pi)$. Waits until $|CTL[L]| > 1$ or $L.status == removed$:
    $oldL \leftarrow LQ.get()$ and $CL.add(oldL)$
    if $L.status\ != removed$:
        $LQ.put(L)$ and $CL.remove(L)$
    $D_i \leftarrow \mathsf{PVSS.PreRecon}(C_i, \mathsf{sk}_i)$
    broadcasts $(\mathsf{recon}, D_i)$.

Step 1. For party $i$, waits until the receipt of $t$ $(\mathsf{recon}, D_i)$,
    $g^s \leftarrow \mathsf{PVSS.Recon}(C, \{D_i\}_{i \in T})$
    $R_e \leftarrow Hash(R_{e-1}, g^s)$
    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 $R_e$
    enters Step 0 to the next round $e+1$.
```

Fig. 3: The consumer process at party i

The consumer process updates the global variables, i.e., L, CL, LQ, CTL 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 $CTL[L].get()$ returns a valid PVSS commitment for every epoch. The fact is guaranteed and it is proved by Lemma 3. The required BRB-like security properties of an epoch are proved by Lemma 4, Lemma 5 and Lemma 6, respectively. Further, we prove the consumer process ensures the BFT security properties [26], [27] by Theorem 3 and Theorem 4.

Note that the gray text in Figure 3 highlights the situation that a faulty party (i.e., the leader L) should be removed. The removal process is introduced in Section IV-D.

*Lemma 3 ($CTL[i]$ **is non-empty**):* The set $CTL[i]$ 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 $CTL[i]$ can be appended without interruption. The condition, i.e., $|CTL[i]| > 1$, in **Step 0** guarantees that $CTL[i]$ is non-empty for all honest parties. However, if party i stops producing valid PVSS commitments in the producer process, $CTL[i]$ will stop growing. $|CTL[i]|$ will be 0 in **Step 0** if i is chosen as leader for $|CTL[i]|$ times. In this case, the consumer process will be suspended and it will be resumed only when i starts to broadcast PVSS commitments again or i is removed by the removal process.

*Lemma 4 (**Validity of each epoch**):* If each non-faulty party i broadcasts D_i in **Step 0**, then all non-faulty parties will accept a beacon value for epoch e.

Proof. Notice that the PVSS commitment to be consumed for current epoch e is the leader's unconsumed earliest proposal. We leverage the mathematical induction to prove the validity of each epoch. In the first epoch 1, the global states are initialized to be consistent for all honest parties. Suppose the global states are consistent for all honest parties and $CTL[L]$ is non-empty (cf. Lemma 3) at the end of epoch $e-1$. We prove that the global states in epoch e in the consumer process are consistent for all honest parties. By Lemma 2, $CTL[L]$ is ordered for all honest parties. So, given the same $v \leftarrow CTL[L].get()$, all non-faulty parties will send (recon, D_i) in **Step 0**. Each party will collect $n - f > t$ valid decryption keys $\{D_i\}$ to obtain g^s by $\mathsf{PVSS.Recon}(C, \{D_i\})$. Then the beacon value R_e is calculated as $R_e = Hash(R_{e-1}, g^s)$. All non-faulty 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 BRB protocol, guaranteeing that all honest parties will receive $2f+1$ $\mathsf{reconReady}$ messages and accept the same beacon value R_e.

Claim 1: 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. Suppose two honest parties i and j send $\mathsf{reconReady}$ message for two different beacon values R_e and R'_e, respectively. Party i must have received a set A of $2f+1$ $\mathsf{reconEcho}$ for R_e and party j must have received a set B of $2f+1$ $\mathsf{reconEcho}$ 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 $\mathsf{reconReady}$ messages for different values, violating the assumption.

*Lemma 5 (**Agreement of each epoch**):* If a non-faulty party accepts R_e, then all non-faulty 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 earliest 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 $t = f+1 > f$, making it impossible for faulty parties to recover the valid secret without participation of an honest party. Further, Claim 1 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 6 (**Termination of each epoch**):* The consumer process eventually outputs a valid beacon value R_e for the epoch e.

Proof. By Lemma 3, $CTL[i]$ 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. Further, with at least $2f+1 > t = f+1$ honest parties, **Step 1** will be eventually invoked. The subsequent steps **Step 2** and **Step 3** follow the same structure as the original BRB protocol. Thus, all honest parties will eventually accept R_e for epoch e. Hence, each epoch, described in a BRB-like protocol, achieves the ***termination*** property.

*Theorem 3 (**Safety of the consumer process**):* For any epoch e, the consumer process will always reach agreement for all honest parties.

Proof. The ***agreement*** of each epoch (cf. Lemma 5) indicates that all honest parties share the same global states at the beginning of each epoch[2]. Hence, each honest party i will broadcast share D_i, guaranteeing that all honest parties will accept the same beacon value R_e due to ***validity*** of each epoch (cf. Lemma 4). Moreover, the consumer process is a combination of infinite epochs. Then, 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. Suppose L is the leader, the $CTL[L]$ is non-empty (cf. Lemma 3) at the beginning of epoch e and the consumer process will eventually output a valid beacon R_e due to the ***termination*** property (cf. Lemma 6). The **Step 0** may suspend if the only one PVSS commitment can be fetched out from $CTL[L]$ and $|CTL[L]| = 1$ holds for a long time. Note that the time is a system parameter, rather than from a global clock. 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**. Therefore, the consumer process is guaranteed to achieve liveness.

*Lemma 7 (**Back-on-track**):* If some party j falls offline or experiences network issues at epoch e', it can get back on track to the latest epoch $e > e'$ by downloading the tuple CTL, CL, LQ, e, R_{e-1} from any honest party.

Proof. If j downloads the tuple from a dishonest party, it will detect inconsistencies with at least $2f+1$ other parties at epoch e. Consequently, j can attempt to download the tuple from another party. Without loss of generality, suppose the honest party i eventually provides correct tuple for j. As designed, CTL stores PVSS commitments for all parties in the producer process. Moreover, elements in $CTL[j]$ are in order (cf. Lemma 2). Although CTL continues to grow as the producer process progresses, this does not impact party j to download earliest PVSS commitment from party i. Besides, the tuple (CL, LQ, e, R_{e-1}) is updated and reaches consensus as consumer ***safety*** (cf. Theorem 3) indicates. Hence, a slow party can always get back on track to the latest epoch e.

[2]It does not mean they share the same state at the same time.

D. The removal process

The removal process is designed to enable the system to remove faulty members without restarting. In the producer-consumer model, each party is designed to produce and consume PVSS commitments in the producer and consumer process, respectively. In case a party i behaves maliciously in the producer process, and all honest parties reach a consensus on "$\perp$". Hence, the length of $CTL[i]$ will not increase, and $CTL[i]$ will eventually become empty after i is elected as leader for $|CTL[i]|$ epochs. Figure 4 depicts the party removal process in a BRB-like protocol, similar to the consumer process where conditions of **Step 2** and **Step 3** are the same with original BRB protocol.

```
Step 0. For party $i$, if $|CTL[L]| = 1$ holds for a period longer than $\Delta$t and no valid receipt (initial, $(L, C, \pi)$) in Step 1 of the producer process. Denote $v = (L, e)$,
    broadcasts (removal, $v$).

Step 1. For party $i$, waits until the receipt of $t$ (removal, $v$),
    broadcasts (removalEcho, $v$).

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

Step 3. For party $i$, waits until the receipt of $2f + 1$ (removalReady, $v$),
    set $L$.status = removed
    notify the consumer process.
    rollback to epoch $v.e$, if needed.
```

Fig. 4: The removal process to remove L at party i

If the leader L's amount of unconsumed commitments at current epoch e is 1, i.e., $|CTL[L]| = 1$, for more than Δt[3] for party i, it broadcasts the initialization message (removal, v), where $v = (L, e)$. Once a party i receives $2f+1$ (removalReady, v), it marks that L is removed and informs its consumer process, which was suspended in **Step 0** in consumer process. Besides, due to termination of BRB(-like) protocols, the removal process might bring fork and honest parties need rollback in **Step 3**. This is discussed by Claim 3.

We prove the ***validity***, ***agreement*** and ***termination*** of the proposed BRB-like protocol by Lemma 8, Lemma 9 and Lemma 10, respectively.

*Lemma 8 (**Validity of the removal process**):* If all honest parties broadcast the removal message at epoch e, then all honest parties will remove L.

Proof. The conclusion is evident, akin to the consumer process (cf. Lemma 4).

Claim 2: No two honest parties will send conflicting message (removalReady, v) and (removalReady, $v' \neq v$).

[3]Note that Δt is a global parameter and it does not require a global clock.

Proof. The proof follows the same reasoning as in Claim 1 and is therefore omitted here.

*Lemma 9 (**Agreement of the removal process**):* If a non-faulty party agrees to remove party L, then eventually all non-faulty parties will agree to remove L.

Proof. By Claim 2, all non-faulty parties will broadcast the $\textsf{removalReady}$ message for the same value v, where $v = (L, e)$. So, if a non-faulty party agrees to remove party L, it must have received $2f + 1$ $(\textsf{removalReady}, v)$ messages, of which at least $f + 1$ came from non-faulty parties. Consequently, all non-faulty parties will broadcast $(\textsf{removalReady}, v)$, either due to seeing the $f + 1$ $\textsf{removalReady}$ messages or due to seeing $2f + 1$ echoes. Claim 2 guarantees that no non-faulty will broadcast $(\textsf{removalReady}, v' \neq v)$. So there will not be $2f + 1$ echoes for v' or $f + 1$ $\textsf{removalReady}$ for v'. Hence, eventually all non-faulty parties will agree to remove party L.

*Lemma 10 (**Termination of the removal process**):* All non-faulty parties will eventually terminates with agreement on whether L is removed or not.

Proof. If L does not provide valid PVSS commitments and it will be removed by honest parties by Lemma 8. Here, we consider the worst-case scenario in the removal process, where a fork may occur among honest parties. Define a set A in which at least one honest party has removed L, another set B containing the other honest parties who have not removed L yet. Due to ***agreement*** of the removal process (cf. Theorem 9), all parties in set B will eventually switch to set A at epoch e^*, which is determined by the last party in set B. e^* is also called the termination epoch, which may not be a deterministic value. We leave it an open question whether $e^* \overset{?}{=} e$ holds forever.

Claim 3: If a fork arises due to the removal process, it will be eventually resolved.

Proof. Suppose e^* to be the determination epoch of the removal process, as defined in Lemma 10. During the period $e^* - e$, at most $e^* - e$ beacon values will have been generated. Since the status of parties in the forking set B will eventually rollback, the fork and the $e^* - e$ associating beacon values will be abandoned. This implies that unpredictability might be affected if $e^* - e > 0$. Summarily, all honest parties will continue the beacon protocol without party L starting from epoch e and share the same global state (CTL, CL, LQ, R_e), even though some parties rely on rollback. Besides, old PVSS commitments in CTL will still be consumed with at least t honest parties and new PVSS commitments will be generated with the remaining $n - 1$ parties' public keys after epoch e.

E. The joining process

To achieve system reconfiguration without the need for restarting, we introduce a party joining process that facilitates the seamless integration of a new party into the system. The joining process is an instance of BRB protocol, hence the properties of ***validity***, ***agreement*** and ***termination*** are guaranteed.

Suppose a joining party θ composes a proposal $v = (e, e^*, \textsf{pk}_\theta, C, \pi)$, where e is the current epoch, $e^* \gg e$ is the expected epoch for θ to appear in the system, $\textsf{pk}_\theta$ is θ's public key, and (C, π) is the output of $\textsf{PVSS.Share}(s, \{\textsf{pk}_i\} \cup \textsf{pk}_\theta)$. Then, θ initiates the BRB protocol to request to join the system by broadcasting $(\textsf{join}, v)$. Denote the termination epoch $e^\#$, which is not a deterministic value. The only distinction with standard BRB protocol (cf. Section II-B) is in **Step 3**, where a party may need to rollback status of (CTL, CL, LQ, R_{e^*}) to the expected epoch e^*. Figure 5 depicts the party joining process using BRB protocol.

```
Step 0. For a joining party $\theta$, invokes $(C, \pi) \leftarrow$
$\textsf{PVSS.Share}$. Set $v = (e, e^*, \textsf{pk}_\theta, C, \pi)$ and
  broadcasts $(\textsf{join}, v)$.

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

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

Step 3. For party $i$, waits until the receipt of
    $2f + 1$ $(\textsf{joinReady}, hv)$, and wait until
    epoch $e^*$ or current epoch $e^\# > e^*$,
  if $e^\# > e^*$:
    rollback $(CTL, CL, LQ, R_{e^*})$ to epoch $e^*$
  $CTL[\theta].put(v)$
  $CL.add(\theta)$
```

Fig. 5: The joining process for party θ at party i

We consider the worst situation where rollback occurs, i.e., $e^\# > e^*$, by Claim 4.

Claim 4: Consider the worst case where an honesty party j accepts θ in **Step 3** at the epoch $e^\# > e^*$. We claim that beacon output will be eventually under consensus in the consumer process, and any fork, if appeared with little probability, will be temporary.

Proof. Normally, all honest parties will put the joining party θ in CTL and CL, since $v.e^* \gg v.e$ for a valid BRB proposal v by θ. Then we consider the worst case. Define a set A of which honest parties that have accepted θ before e^*, and another set B of honest parties that have not. Obviously, $|A| + |B| = 2f + 1$. ($|A|$ can be 0 in our design.) Due to the BRB ***agreement*** and ***termination*** properties, parties in B will eventually rollback to epoch e^* and align their status with those in A. Summarily, all honest parties eliminate the fork and share the same state in epoch e^*, though some honest parties rely on rollback. Consequently, starting from epoch e^*, newly generated PVSS commitments in the producer process will include the public key of θ. Besides, all honest parties will update (CTL, CL, LQ, R_{e^*}) to the same value and run the consumer process with participation of θ.

Further, we discuss the impact to unpredictability of the beacon. When a party j in B rolls back, we consider that party j is changed from B to A. Denote $e^\# > e^*$ as the

BRB termination epoch, determined by the last party in B. Hence, at most $e^{\#}-e^{*}$ beacon values may have been generated with $e^{\#} - e^{*}$ PVSS commitments being decrypted in the consumer process. This implies that unpredictability may be compromised for a maximum of $e^{\#} - e^{*}$ epochs.

F. *Summary of the processes*

Bracha Reliable Broadcast (BRB) or BRB-like protocol is a one-shot communication abstraction to achieve consensus. The original BRB protocol has the security properties of ***validity***, ***agreement*** and ***Termination*** (cf. Section II-B). We also define and prove the security properties for the BRB-like protocols when they are designed in the consumer and removal processes. To provide a concise overview of the processes, Table III is presented. The column **Instance(s)** indicates how many BRB(-like) protocols is 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 III: Summary of the processes in $AsyRand$

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

The producer and joining processes are directly leverage the BRB protocol. All parties in the producer process, as well as the joining node in the joining process, first invoke the PVSS.Share algorithm and initiate the BRB 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 BRB-like protocols. In the consumer process, the PVSS.PreRecon and PVSS.Recon algorithms are executed to recover the leader's secret. Both the producer and joining processes run with an infinite number of BRB(-like) instances, which form the foundation of $AsyRand$. The removal and joining processes require only a single instance of the BRB(-like) protocol, achieving reconfiguration.

V. Properties of $AsyRand$

In this section, we argue the security properties of $AsyRand$, focusing on liveness, bias-resistance, unpredictability, public verifiability, guaranteed output delivery and responsiveness.

*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 IV. In the producer process, each party i continuously invokes the BRB protocol to commit new PVSS commitments. The PVSS commitments are verified as valid by all honest parties or any external user. Hence, a valid PVSS commitment can always reach consistency among honest parties. By Theorem 2, we prove that no adversaries can prevent new PVSS commitments from being appended into global state CTL for honesty parties. In the consumer process, parties collectively consume the previously PVSS commitments stored in CTL. Lemma 3 has shown that $CTL[L]$ is not empty before L is elected as leader. Thus, the leader L's earliest PVSS commitment, i.e., $v \leftarrow CTL[L].get()$, can always be available for each epoch e. By Theorem 4, we prove that no adversaries could prevent a PVSS commitment from being consumed. If $CTL[L]$ remains empty for a long time, party L will be removed in the removal process by all honest parties. Both the removal and joining processes will eventually reach agreement due to BRB(-like) **agreement** and **termination** properties. Thus, adversaries cannot prevent the processes of $AsyRand$ from proceeding in the proposed producer-consumer model.

*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 $|CTL[L]| > 0$ holds (cf. Figure 3) for leader L when the earliest PVSS commitment $v \leftarrow CTL[L].get()$ 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, $R_e = Hash(g^s, R_{e-1})$, depends on the secret g^s 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, g^s 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 g^s and the previous beacon value R_{e-1}. We have proved that the proposed PVSS scheme guarantees ***IND1-Secrecy*** in Appendix C, ensuring 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 [13]. 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 among the parties, making them publicly verifiable with the PVSS.Verify algorithm. In the consumer process, the decrypted PVSS shares, obtained using the PVSS.PreRecon algorithm, are broadcast by all parties. 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 (**Guaranteed Output Delivery**):* A new beacon value is guaranteed to be output in each epoch e.

$Proof$. By Lemma 3, the PVSS commitment queue $CTL[L]$ of the leader L is always non-empty during the consumer process. Hence, a secret g^s hidden in $v \leftarrow CTL[L].get()$ can be eventually recovered with at least $n - f > t > f$ PVSS decrypted shares from honest parties. Further, a new beacon value R_e for epoch e is guaranteed to be calculated by $Hash(g^s, R_{e-1})$.

*Theorem 10 (**Responsiveness**):* $AsyRand$ is responsive, meaning that beacon values are delivered at the speed of the real network.

$Proof$. Both the producer and the consumer processes of $AsyRand$ are depicted in an asynchronous setting. The objects produced and consumed in $AsyRand$ are PVSS commitments contributed by all parties. During the producer process, PVSS commitments are reliably generated as parties independently invoke the PVSS.Share algorithm and reach consensus on each commitment using BRB protocols. In the consumer process, honest parties output the random beacon value R_e for each epoch e using the leader's earliest PVSS commitment in a BRB-like protocol. Furthermore, Lemma 7 demonstrates that a slow party can get back on track to the latest epoch in the consumer process. This implies that $AsyRand$ delivers beacon values at the pace of the consumer process, advancing at the speed of the actual asynchronous network.

Functionality The proposed PVSS scheme

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

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

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

$$p(x) \xleftarrow{R} poly(\cdot), \text{where } p(0) = s$$
$$C = \left\{ \{C_i = \mathsf{pk}_i^{p(i)}\}_{i \in [1, \cdots, n]} \right\}$$
$$p'(x) \xleftarrow{R} p(x), \text{where } \ p'(0) = s' \xleftarrow{R} \mathbb{Z}_p$$
$$C' = \left\{ \{C'_i = \mathsf{pk}_i^{p'(i)}\}_{i \in [1, \cdots, n]} \right\}$$
$$\pi \leftarrow \begin{cases} C', \\ c = H(C, C'), \\ \tilde{s} = s' - cs, \\ \{\tilde{p}(i) = p'(i) - c \cdot p(i)\}_{i \in [1, \cdots, n]}, \end{cases}$$

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

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

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

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

$g^s \leftarrow \mathsf{PVSS.Recon}(C, \{D_i\}_{i \in T})$:

$$e(D_i, \mathsf{pk}_i) \stackrel{?}{=} e(g, C_i)\ \forall i \in T$$
$$\mu_i = \textstyle\prod_{j \in T, j \neq i} \frac{j}{j-i}$$
$$\textstyle\prod_{i \in T} D_i^{\mu_i} = \prod_{i \in T} g^{\mu_i \cdot p(i)} = g^{p(0)} = g^s$$

Fig. 6: Construction of the proposed PVSS scheme

VI. New PVSS construction

Figure 6 depicts the concrete construction of the proposed PVSS scheme using Σ protocol and NIZK proof (cf. Appendix B). Particularly, C' is the commitment, c is the challenge, $(\tilde{s}, \tilde{p}(i))$ is the response in PVSS.Share. The intpl algorithm in PVSS.Verify is the Lagrange interpolation. The required security properties are proved in Appendix C. The complexity and the performance comparison with state-of-the-art works are presented in Appendix D.

VII. Implementation and evaluation

The modified BRB protocol (cf. Section II-B) 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, as Figure 3 shows. Consequently, the complexity of the consumer process is $O(n^2)$. Therefore, the overall communication complexity of $AsyRand$ is $O(n^2)$.

We implement the proposed PVSS scheme with Charm-Crypto library, which is a framework for constructing cryptographic schemes. We choose an asymmetric curve *MNT159* to implement the proposed PVSS scheme. For the network setup, we implement a fully-connected peer-to-peer (p2p) network based on TCP socket programming. 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 and runs Linux ubuntu-bionic-18.04 with Python 3.6.9. The proof-of-concept implementation is available on Github[4].

TABLE IV: 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 IV. It is evident that the bilinear pairing ($e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$) and $\text{Exp}_{\mathbb{G}_1}$ incur a significantly higher cost than $\text{Exp}_{\mathbb{G}_0}$.[5] We then assess the performance of our proposed PVSS scheme in comparison with related PVSS schemes in Appendix D.

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

[4]https://github.com/AppCrypto/beacon

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

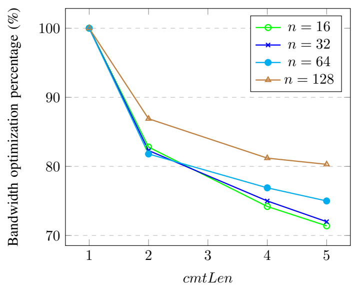

Fig. 7: Bandwidth optimization with multiple PVSS commitments broadcast

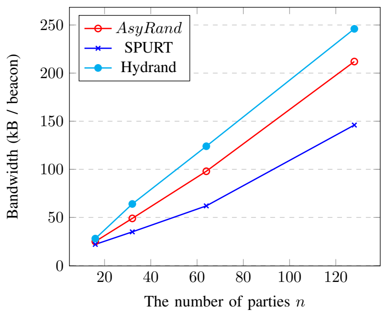

Fig. 8: Bandwidth per beacon

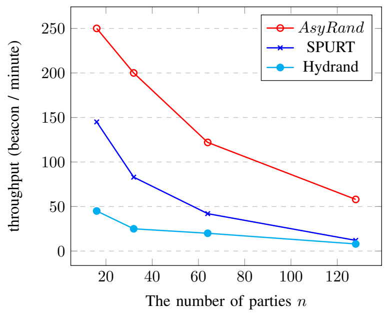

Fig. 9: Average beacon throughput per minute

process allows parties to broadcast an arbitrary number of PVSS commitments independently.

As expected, the producer process operates faster than the consumer process. By Theorem 10, it is shown 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 *bufferLen*, the number of unconsumed PVSS commitments from a party, and $cmtLen$, the number of PVSS commitments sent at a time by a party. Adjusting *bufferLen* effectively slows down the producer process, while modifying $cmtLen$ reduces bandwidth consumption, as multiple PVSS commitments can be transmitted via a single BRB protocol.

By setting *bufferLen* to 2 or 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 7 shows the results of bandwidth optimization. Figure 8 illustrates the bandwidth usage (both sent and received data) per beacon output. For $n = 32$ with *bufferLen* $= 2$ and $cmtLen = 4$, the average bandwidth of a party is around 51kB per beacon. The bandwidth usage in $AsyRand$ is higher than in SPURT (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 in Figure 9, $AsyRand$ achieves significantly higher throughput compared to Hydrand and SPURT. Specifically, with $n = 128$, $AsyRand$ produces 58 beacons per minute, while SPURT and Hydrand produce only 12 and 8 beacons per minute, respectively.

VIII. Concluding remarks

In this paper, we introduce $AsyRand$, a distributed randomness beacon protocol designed for asynchronous settings and based on a novel publicly verifiable secret sharing (PVSS) scheme. $AsyRand$ operates within a producer-consumer model. The producer process generates PVSS commitments, which are then used by the consumer process to continuously produce beacon values. To ensure consensus and consistency among distributed parties, we integrate Bracha Reliable Broadcast (BRB) or BRB-like protocols. Additionally, $AsyRand$ supports reconfiguration, enabling the removal of faulty parties and the addition of new ones without compromising the protocol's eventual consensus. We offer proofs showing that these reconfiguration processes do not impact the consistency of the beacon protocol. Our analysis also establishes that $AsyRand$ achieves key properties, including liveness, unpredictability, bias-resistance, public verifiability, guaranteed output delivery and responsiveness. Experimental results further validate the feasibility and effectiveness of $AsyRand$. In the future, we plan to explore verifiable batched secret sharing [43] to further reduce communication overhead and improve throughput. Additionally, batched secret sharing could be leveraged to achieve complete unpredictability while maintaining state-of-the-art complexity as $AsyRand$.

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] Interoperable Randomness Beacons. https://www.nist.gov/programs-projects/nist-randomness-beacon, Accessed: 2024-07-06.
[11] Choi, K., Manoj, A., & Bonneau, J. SoK: Distributed randomness beacons. In *S&P*, 2023, pp. 75-92.
[12] Bonneau, J., Clark, J., & Goldfeder, S. On bitcoin as a public randomness source. http://eprint.iacr.org/2015/1015, 2015.
[13] Schindler, P., Judmayer, A., Stifter, N., & Weippl, E. HydRand: Efficient Continuous Distributed Randomness. In *S&P*, 2020, pp. 73-89.
[14] 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.
[15] Bünz, B., Goldfeder, S., & Bonneau, J. Proofs-of-delay and randomness beacons in Ethereum. In *S&B*, 2017.

[16] Cascudo, I., & David, B. SCRAPE: Scalable Randomness Attested by Public Entities. In *ACNS*, 2017, pp. 537-556.
[17] Cascudo, I., & David, B. ALBATROSS: Publicly Attestable Batched Randomness Based on Secret Sharing. In *ASIACRYPT*, 2020, pp. 311-341.
[18] 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.
[19] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP*, 2017, pp. 51-68.
[20] Hanke, T., Movahedi, M., & Williams, D. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.
[21] Cascudo, I., David, B., Shlomovits, O., & Varlakov, D. Mt. Random: Multi-tiered randomness beacons. In *ACNS*, 2023, pp. 645-674.
[22] 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.
[23] Boneh, D., Bonneau, J., Bünz, B., & Fisch, B. Verifiable delay functions. In *Crypto*, 2018, pp. 757-788.
[24] Schoenmakers, B. A Simple Publicly Verifiable Secret Sharing Scheme and Its Application to Electronic Voting. In *CRYPTO*, 1999, pp. 148-164.
[25] 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.
[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] Wang, X., Duan, S., Clavin, J., & Zhang, H. BFT in blockchains: From protocols to use cases. *ACM Comput. Surv.*, 2022, 54(10s), 1-37.
[29] Bracha, G. An asynchronous [(n-1)/3]-resilient consensus protocol. In *PODC*, 1984, pp. 154–162.
[30] Damgård, I. On Σ-Protocols, https://www.cs.au.dk/~ivan/Sigma.pdf, Accessed: 2024-07-06
[31] Fiat, A., & Shamir, A. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, 1986, pp. 186–194.
[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] Stadler, M. Publicly verifiable secret sharing. In *Eurocrypt*, 1996, pp. 190-199.
[41] Cascudo, I. and David, B. Publicly verifiable secret sharing over class groups and applications to DKG and YOSO. In *Eurocrypt*, 2024, pp. 216-248.
[42] 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.
[43] 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.

Liang Zhang obtained excellent programming skills in Baidu Co., Ltd. after acquiring bachelor's degree from Huazhong University of Science and Technology. He then acquired Ph.D. degree from Fudan University in 2022. He is currently a postdoctoral fellow at Hong Kong University of Science and Technology. His research interests include blockchain and applied cryptography. His recent publications are included in *IEEE TIFS*, *IEEE TC*, *IEEE TSC*, *IEEE TBD*, *CJ*, *CSCWD'25*, *APWeb-WAIM'23*, *CF'22* and *SACMAT'22*.

Tao Liu was born in 2002. He joined Hainan University in September 2021. He is currently pursuing a bachelor's degree at the School of Cyberspace Security (School of Cryptography) at Hainan University and will pursue a master's degree at the School of Cyber Science and Engineering, Wuhan University, starting from September 2025. His research interests include blockchain and applied cryptography.

Zhanrong Ou was born in 1998. He received a bachelor's degree in Hanshan Normal University, Chaozhou, Guangdong province, in 2021. He joined Hainan University in September 2022. Now, he is pursuing his master's degree in the School of Cyberspace Security (School of Cryptology) at Hainan University. His research interests consist of blockchain and decentralized randomness beacon.

Haibin Kan was born in 1971. He received a Ph.D. from Fudan University, Shanghai, China, 1999. From June 2002 to February 2006, he was with the Japan Advanced Institute of Science and Technology as an assistant professor. He went back Fudan University in February 2006, where he is currently a full professor. He is also the Director of the Shanghai Blockchain Engineering Research Center. His research interests include coding theory, cryptography, and computation complexity.

Jiheng Zhang received both Ph.D. and M.S. degrees in operations research from Georgia Institute of Technology, MS degree in mathematics from the Ohio State University, and BS degree in mathematics in Nanjing University. He joined Industrial Engineering and Decision Analytics (IEDA) of the Hong Kong University of Science and Technology (HKUST) as an Assistant Professor in 2009, and was promoted to Associate Professor in 2015 and then to full Professor in 2020. He now is the department head of IEDA of HKUST. He is also the associate director of HKUST Crypto-Fintech Lab.

Appendix

A. *Benefits of the producer-consumer model*

The producer-consumer model offers below advantages.

- The producer and consumer processes in each party can work simultaneously and they are coordinated by the queue. If queue is non-empty, a beacon value is guaranteed to be delivered. Besides, no global clock is required, as a consumer process can suspend and wait until new PVSS commitment is put into the queue. Consequently, the producer-consumer model ensures asynchrony in local processes, while the BRB protocol achieves asynchrony across all parties.
- The model enables non-faulty parties to set a default agreement of "$\perp$" in the producer process when a party acts as a PVSS dealer and it is unknown to be faulty or not. "$\perp$" means that no honest party takes the action of putting PVSS commitment into the queue. Further, a party can simply detect whether another party is faulty/malicious by monitoring queue length of the party. If the length is 1 for a long time, honest parties can start to remove the party using a BRB-like protocol before consuming its last PVSS commitment. This approach allows a party to detect faulty behaviors without relying on detailed messages in the BRB or BRB-like protocols. Additionally, this faulty detection method allows parties to trigger the removal process without system restart.
- The epoch sequence number defined in the consumer process defines a "chronological clock" for all parties, similar to block height in blockchain systems. This feature facilitates the removal and joining processes with unified and determined global epoch numbers.
- This model decouples the production and consumption of PVSS commitments, making it convenient to identify performance bottlenecks in the beacon execution and facilitating further optimization of throughput and bandwidth.

B. *Σ-protocol and NIZK proof*

In a Σ-protocol [30], 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 Σ-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 Σ-protocol by leveraging Fiat-Shamir heuristic [31], where the challenge value e is calculated by $\mathcal{P}$ with random oracle. Σ-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.

C. *Security analysis of the proposed PVSS*

By Theorem 11, Theorem 12 and Theorem 13, we prove the required security properties, defined in Section II-A, of the proposed PVSS scheme.

*Theorem 11 (**Correctness**):* Given at least t decrypted shares $\{D_i\}_{i \in T}$, the secret g^s can be successfully recovered.

Proof. The polynomial property ensures that $\sum_{i \in T} \mu_i \cdot p(i)$ interpolates to the value $p(0)$, where $\{\mu_i\}$ are the lagrange coefficients. Consequently, the PVSS secret g^s 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 11 (Σ **Completeness**):* If the dealer knows the witness g^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 witness g^s, he can prove it in (C, π) with probability 1.

*Lemma 12 (Σ **Soundness**):* If the dealer does not know g^s, he cannot cheat the verifier successfully.

Proof. It is widely adopted to prove Σ-protocol ***Soundness*** by extracting the witness when two accepting proofs with the same commitment and different challenges are given [30]. 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 witness g^s can be extracted with (C, π_1, π_2).

*Theorem 12 (**Public Verifiability**):* The messages from the dealer and the shares sent by shareholders are publicly verifiable.

Proof. By Lemma 11 and Lemma 12, the Σ-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 $\mathsf{PVSS.Share}$ algorithm represents a non-interactive proof of knowledge for the witness g^s. 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 $e(D_i, \mathsf{pk}_i) \overset{?}{=} e(g, C_i)$, as shown in the $\mathsf{PVSS.Recon}$ algorithm.

*Lemma 13 (Σ **Zero knowledge**):* The output of $\mathsf{PVSS.Share}$ reveals no information about a shareholder P_i's $g^{p(i)}$ or the dealer's witness g^s.

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 $\mathsf{PVSS.Share}$ 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 witness $g^{p(i)}$ via $p(i)$.

Besides, it is also infeasible to defer P_i's witness $g^{p(i)}$ using $C_i' = \mathsf{pk}^{p(i)} = (g^{p(i)})^{\mathsf{sk}_i}$, due to discrete logarithm problem.

*Definition 1 (**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 $\mathsf{PVSS.Setup}$ algorithm and sends $(g, \mathsf{pk}_i, \mathsf{sk}_i)$ to each uncorrupted shareholder P_i.
2) $\mathcal{C}$ sends public information and $t-1$ corrupted private keys $\{\mathsf{sk}_i\}$ to $\mathcal{A}$.
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 $\mathsf{PVSS.Share}$ algorithm with secret x_b and sends all the output to $\mathcal{A}$, along with x_{1-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 13 (**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 g^s.

Proof. By Lemma 13, the Σ-protocol used in the proposed PVSS provides zero knowledge about each shareholder's witness $g^{p(i)}$ for $i \in [1, \cdots, n]$ in the $\mathsf{PVSS.Share}$ 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 1 based on discrete logarithm (DL) assumption. Without lose of generality, denote the first $t-1$ shareholders, i.e., $[P_1, \cdots, P_{t-1}]$, as the corrupted parties.

1) The challenger $\mathcal{C}$ runs the $\mathsf{PVSS.Setup}$ algorithm and sends $(g, \mathsf{pk}_i, \mathsf{sk}_i)$ to each uncorrupted shareholder $P_i \in [t, \cdots, n]$.
2) $\mathcal{C}$ sends public information $(g, \{\mathsf{pk}_i\}_{i \in [1, \cdots, n]})$ and corrupted private keys $\{\mathsf{sk}_i\}_{i \in [1, t-1]}$ to $\mathcal{A}$.
3) For $1 < i \leq t-1$, $\mathcal{C}$ chooses uniformly random values $s_i \leftarrow \mathbb{Z}_p$ and sets $C_i = \mathsf{pk}_i^{s_i}$. For $t \leq i \leq n$, $\mathcal{C}$ generates $C_i = \mathsf{pk}_i^{p(i)}$ where $p(x)$ is the (t-1)-degree polynomial ensuring the PVSS secret $x_b = g^{p(0)}$ and $p(j) = s_j$ for $1 \leq j \leq t-1$. Denote $C = \{C_i\}_{i \in [1, \cdots, n]}$. It calculates the NIZK proofs π for C, as the dealer does. Finally, $\mathcal{C}$ sends (C, π) to $\mathcal{A}$.
4) $\mathcal{A}$ makes a guess of b'.

The information sent to $\mathcal{A}$ is identical to the output of the $\mathsf{PVSS.Share}$ algorithm. With $t-1$ shareholders corrupted, $\mathcal{A}$ can interpolate $g^{p'(0)}$ using each $g^{p(i)}$, where $p'(x)$ is a (t-2)-degree polynomial. It follows that $p(x) = p'(x) + \tau \prod_{j=1}^{t-1}(x-j)$, where τ is an unknown value randomly distributed in $\mathbb{Z}_p$. Hence, $g^s = g^{p(0)} = g^{p'(0) + \tau \prod_{j=1}^{t-1}(0-j)}$ is also a random value in $\mathbb{G}$. Thus, $\mathcal{A}$ can successfully guess $b' = b$ with probability $1/2 + \epsilon$, where ϵ is the probability to break DL assumption. Hence, $|\Pr[b = b']\text{-}1/2| = \epsilon$ which is negligible.

D. PVSS Comparison

SCRAPE [16] is notable for being the first PVSS scheme with $O(n)$ verification complexity, leveraging Reed-Solomon codes. ALBATROSS [17], building on SCRAPE [16], introduces packed (or ramp) shamir secret sharing in PVSS and linear t-resilient function. In YOLO YOSO [18], the authors introduce two PVSS protocols (i.e,. HEPVSS and DHPVSS) based on SCRAPE, where specific techniques are applied to generating proofs of (re-)share validity. More recently, qCLPVSS [41] is introduced, based on DHPVSS, to recover the original secret on $\mathbb{Z}_p$, making it applicable in DKG and YOSO MPC. The underlying mathematical technique depends on class group. while the complexity is similar to DHPVSS [18].

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

TABLE V: Computation complexity

Ref.	**Sharing**	**Verfication**		**Reconstruction**	
	Exp.	Exp.	Pair	Exp.	Pair
$[16]_{DBS}$	$2n$	n	$2n$	$t+1$	$2t+1$
$[16]_{DDH}$	$4n$	$5n$	$-$	$5t+3$	0
$[17]_{ALBAT.}$	$2n$	$2n$	$-$	$6t$	$-$
$[18]_{HEPVSS}$	$7n$	$4n$	$-$	$3t$	$-$
$[18]_{DHPVSS}$	$n(n-t+2)+2$	$n(n-t)+4$	$-$	$5t$	$-$
Ours	$2n$	$2n$	$-$	t	$2t$

Communication Complexity: In the sharing phase, the dealer publishes the encrypted shares C and the corresponding NIZK

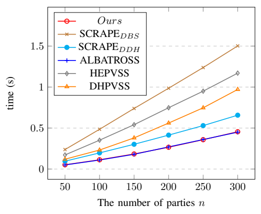

Fig. 10: Comp. cost in the sharing

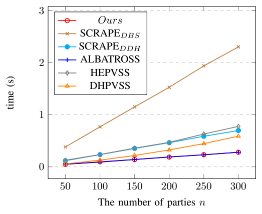

Fig. 11: Comp. cost of in the verification

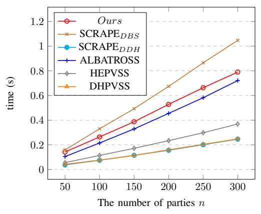

Fig. 12: Comp. cost in the reconstruction

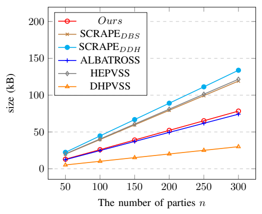

Fig. 13: Comm. cost in the sharing

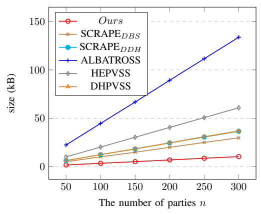

Fig. 14: Comm. cost in the reconstruction

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 $\mathbb{Z}$. 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 g^s.

TABLE VI: Communication complexity

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

By Table V and Table VI, we compare the complexity between our proposed PVSS scheme and previous schemes, where n is the number of shareholders and t is the threshold value.

Figures 10, 11, and 12 illustrate the computation costs for sharing, verification, and reconstruction phases, respectively. Since $\mathbb{G}_1$ (cf. Exp$_{\mathbb{G}_1}$ in Table IV) is used in SCRAPE$_{DBS}$ [16], it results in the highest computation overhead in the distribution phase. The results also suggest that ALBATROSS [17], which builds upon SCRAPE$_{DDH}$ [16], shifts the computational burden from the sharing and verification phases to the reconstruction phase. The superlinear costs in sharing (cf. Figure 10) and verification (cf. Figure 11) 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 12), primarily because they use bilinear pairings instead of DLEQ for verifying decrypted PVSS shares.

Figures 13 and 14 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.