SCIENCE CHINA
Information Sciences

• RESEARCH PAPER •

February 2025, Vol. 68, Iss. 2, 122304:1–122304:16
https://doi.org/10.1007/s11432-024-4232-7

PltcRB: a practical distributed randomness beacon with optimal amortized communication complexity

Zheyi WU, Haolin LIU & Lei WANG*

School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, Shanghai 200240, China

Received 5 March 2024/Revised 2 August 2024/Accepted 28 November 2024/Published online 16 January 2025

Abstract The distributed randomness beacon (DRB) is a crucial tool for continuously generating unpredictable, bias-resistant, and publicly verifiable random numbers on a regular basis. This is particularly useful for applications such as lotteries, electronic voting, and cryptographic parameter generation. However, existing studies either require complex communication or rely on a public bulletin board to meet security requirements. This brings a performance bottleneck when dealing with a large number of participants. This paper introduces a novel DRB protocol PltcRB based on timed commitments, eliminating the need for a public bulletin board. Our approach achieves optimal communication complexity of $O(n)$ while maintaining the desired properties of a DRB protocol. The computation complexity is also $O(n)$ when n is larger than the security parameter κ. These results demonstrate the practicality and performance of PltcRB, supported by our experimental analysis under various configurations.

Keywords timed commitment, randomness beacon, random number, optimal communication complexity, permissionless setting

Citation Wu Z Y, Liu H L, Wang L. PltcRB: a practical distributed randomness beacon with optimal amortized communication complexity. Sci China Inf Sci, 2025, 68(2): 122304, https://doi.org/10.1007/s11432-024-4232-7

1 Introduction

Verifiable random numbers have been widely used in many practical applications. Examples include anonymous communication [1–3], online gaming and gambling [4], electronic voting [5], cryptographic parameter generation [6, 7], and publicly auditable auctions [4]. Notably, permissionless blockchain has become an important research field in recent years, and generating a verifiable random number is essentially crucial for blockchain leader selection [8,9], blockchain sharding [10–12], and smart contracts [13,14]. The security of these applications highly relies on public availability, bias-resistance, and unpredictability of the underlying verifiable random numbers. Thus, it leads to an important and long-term research topic: how to generate publicly verifiable and trustworthy random numbers, among a set of mutually untrustworthy parties, particularly in the permissionless setting?

A seminal solution proposed by Rabin [15] is a randomness beacon. Its functionality is continuously producing bias-resistant, unpredictable, and publicly verifiable random numbers on a regular basis. Informally, a randomness beacon protocol should satisfy the following properties: liveness means that the protocol can proceed when an adversary exists; bias-resistance ensures that the adversary cannot influence the output values with its strategy; unpredictability requires that no one can predict future random numbers and public verifiability guarantees that a third party can verify the correctness of the outputs.

Such a protocol is usually designed round-based, where a verifiable random number is output at the end of every round. Hence, we model the randomness beacon as iterating a verifiable function $\mathcal{F}$ sequentially in this paper. More specifically, at round i the execution is denoted as $(R_i, \pi_i) \leftarrow \mathcal{F}(i, R_{i-1}; r_i)$, where R_{i-1} is the random number output at the round $i-1$, r_i is an auxiliary random number at the round i, R_i is the random number output at the round i, and π_i is a proof of the execution at the round i. Importantly, π_i must be publicly verifiable to ensure the correctness of the execution of $\mathcal{F}$ during round i, and hence the trustworthiness and availability of R_i.

Ever since proposed and formalized, many researchers have been making significant efforts to devise various randomness beacon protocols (e.g., [8, 16–19]). As a high-level summary, these protocols are

* Corresponding author (email: wanglei_hb@sjtu.edu.cn)

divided into two categories depending on whether a third trusted party is used. Intuitively, it is relatively easier to build an efficient randomness beacon with the aid of a trusted third party. Indeed, several practical randomness beacons adopt a trusted third party, such as NIST (National Institute of Standards and Technology) randomness beacon [16] and Random.org. Such designs suffer the risk of a single point of failure in practical usage. More seriously, the extent of the trust in a third party for the randomness beacon is too strong and rather unrealistic, particularly in the permissionless setting.

This paper is interested in building a randomness beacon without the aid of a trusted third party. Typically, such a randomness beacon is designed and maintained by a set of parties in a distributed manner, referred to as distributed randomness beacon (DRB). These parties are mutually untrustworthy, and malicious parties intend to attack the protocol arbitrarily. Therefore, ensuring robustness is challenging for DRB protocol. This is commonly solved in the field of threshold cryptography, which enables the reconstruction of a secret value through exchanging distributed secret shares. More specifically, three main techniques have been introduced to developing DRB protocols, i.e., threshold signature (TS) [20–23], threshold encryption (TE) [24, 25], and publicly verifiable secret sharing (PVSS) [17–19, 26–29].

Cachin et al. [20] proposed the first TS-based DRB in 2005, where all participants create their signature shares and construct the same beacon output after obtaining more than the threshold number of shares. Later in 2020, Galindo et al. [23] constructed a TS-based DRB, which continuously signs a predetermined public message in a threshold manner to generate random values. Moreover, another TS-based randomness beacon Drand [22] is used as a public service, and the blockchain Dfinity [21] uses TS to generate random values.

HERB [24] uses threshold ElGamal encryption to build a DRB. Every participant publishes a ciphertext on a public bulletin board (PBB). Then these ciphertexts are aggregated into one ciphertext via cryptographic homomorphic operations. The decryption of the aggregated ciphertext requires at least a threshold number of participants to collaborate. The resulting plaintext will be the output of the protocol. Nguyen-Van et al. [25] added a trust third party to HERB, which collects and decrypts the ciphertext.

SCRAPE [18] proposed the first PVSS-based DRB protocol. All nodes publish the encrypted shares on a PBB, then each node reveals its secret value. If one node fails or refuses to do it, other nodes can reconstruct its secret collectively. The output random number is finally computed from all secret values. In 2017, RandHound [17] and RandHerd [17] improved the scalability by dividing nodes into small groups of size c.

In 2020, HydRand [19] reduces the communication complexity to $O(n^2)$ and does not require a PBB. In 2021, GRandPipper [27] improves the security bound of HydRand to $1/2$ while keeping the communication complexity as $O(n^2)$. BRandPipper [27] solves the unpredictability problem of GRandPipper with a communication complexity of $O(n^3)$. In 2022, Spurt [28] aggregates shares more than the threshold number into one and achieves perfect unpredictability, in a partially synchronous network. Spurt uses a state machine replication protocol to avoid the usage of the broadcast channel. In 2023, OptRand [29] tolerates $1/2$ faulty nodes in a synchronized network. It proceeds at the latency of the network in the optimal case when the leader is honest and faulty nodes are no more than $1/4$.

From a high-level overview, all the above DRB protocols use threshold techniques. As a result, they require interactive communication among parties for secret reconstruction, causing the communication complexity to be at least $O(n^2)$. Such communication complexity will be a serious bottleneck problem in a large scale of parties, e.g., in the permissionless setting.

Thus, it is important to design a DRB protocol with a low communication complexity. One solution is to use time-related cryptographic primitives, such as verifiable delay function (VDF) [30] and timed commitment (TC) [31]. These two primitives have a common property that the nodes can obtain the secret values through local computation. Therefore, they do not need to communicate with each other to reconstruct the secret values, unlike when using threshold techniques. As a trade-off, these two primitives bring additional computation complexity.

For a given input, the output of the corresponding VDF cannot be obtained until a predefined amount of time-consuming computation is performed. Hence the output is kept secret until a future time point. In 2015, Unicorn [7] was proposed based on a VDF. The parties publish their entropy contributions directly on the bulletin board during the specified period. These entropy contributions are used as the input of a VDF, and the random number is the output of the VDF. In 2021, RandRunner [32] was proposed based on trapdoor VDFs. The parties take turns as round leaders to compute its VDF to generate a random value. With the aid of trapdoor, the execution is highly efficient, when the round

Table 1 Comparison of DRB. Crypto. Primitives: cryptographic primitives; Comm. Comp.: communication complexity; Comp. Comp.: computation complexity; Verif. Comp.: verification complexity; Comm. Model: communication model; Fault Tol.: fault tolerance; Adv. Attack: adversary attack. $\triangleright$ Recovery cost denotes the communication cost for recovering the message from the malicious behavior. $\triangleleft$ $\bullet$ denotes the usage of a PBB and $\circ$ denotes not. $\sharp$ Adversary attack denotes the worst-case scenario an adversary can exploit to attack the protocol. $\star$ The computation complexity is related to the time parameter of TCs. $\diamond$ The recovery requires local computation. $\ddagger$ c denotes the size of the committee in RandHound and RandHerd.

Protocol	Crypto. Primitives	Comm. Comp.	Comp. Comp.	Verif. Comp.	Recovery cost$^{\triangleright}$	PBB$^{\triangleleft}$	Comm. Model	Fault Tol.	Adv. Attack$^{\sharp}$
SCRAPE [18]	PVSS	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^3)$	$\bullet$	Sync.	1/2	Bias
HERB [24]	HE	$O(n^2)$	$O(n)$	$O(n)$	$O(n^3)$	$\bullet$	Sync.	1/3	Bias
Bicorn [35]	TC	$O(n)$	$O(n)^{\star}$	$O(n)$	$O(1)^{\diamond}$	$\bullet$	Sync.	1/2	Predict
HeadStart [33]	VDF	$O(n\log n)$	VDF	$O(\log n)$	$O(1)^{\diamond}$	$\bullet$	Sync.	1/2	Predict
Unicorn [7]	VDF	$O(n)$	VDF	$O(n)$	$O(1)^{\diamond}$	$\bullet$	Async.	1/2	Predict
Dfinity [21]	TS	$O(n^2)$	$O(n)$	$O(1)$	$O(n^3)$	$\bullet$	Sync.	1/2	Bias
RandHound [17]	PVSS	$O(c^2n)^{\ddagger}$	$O(c^2n)$	$O(c^2n)$	$O(n^3)$	$\circ$	Sync.	1/3	Bias
RandHerd [17]	TS	$O(c^2\log n)^{\ddagger}$	$O(c^2\log n)$	$O(1)$	$O(n^4)$	$\circ$	Sync.	1/3	Bias
HydRand [19]	PVSS	$O(n^2)$	$O(n)$	$O(n)$	$O(n^2)$	$\circ$	Sync.	1/3	Bias
GRandPiper [27]	PVSS	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^3)$	$\circ$	Sync.	1/2	Bias
BRandPiper [27]	VSS	$O(n^3)$	$O(n^2)$	$O(n^2)$	$O(n^4)$	$\circ$	Sync.	1/2	Bias
Spurt [28]	PVSS	$O(n^2)$	$O(n)$	$O(n)$	$O(n^3)$	$\circ$	Psync.	1/3	Bias
OptRand [29]	PVSS	$O(n^2)$	$O(n)$	$O(n)$	$O(n^3)$	$\circ$	Sync.	1/2	Bias
Cachin et al. [20]	TS	$O(n^2)$	$O(n)$	$O(1)$	$O(n^3)$	$\circ$	Sync.	1/2	Predict
DRAND [22]	TS	$O(n^2)$	$O(n)$	$O(1)$	$O(n^3)$	$\circ$	Sync.	1/2	Predict
RandRunner [32]	VDF	$O(n^2)$	VDF	$O(1)$	$O(1)^{\diamond}$	$\circ$	Sync.	1/2	Predict
PltcRB	TC	$O(n)$	$O(n)^{\star}$	$O(1)$	$O(1)^{\diamond}$	$\circ$	Sync.	1/3	Bias

leader is honest. Otherwise, other parties can independently compute the round output after performing amounts of computations. HeadStart [33] was proposed in 2022 and it is similar to Unicorn: a PBB is used to collect randomness and a VDF is evaluated based on collected randomness to produce the result of the protocol. HeadStart utilizes the Merkle tree to reduce the verification complexity.

Notably, while the number of communication messages reduces, the usage of reliable broadcast in RandRunner [32] introduces $O(n^2)$ communication complexity. The size of messages in HeadStart [33] grows to $O(\log n)$, resulting in communication complexity $O(n\log n)$.

A TC introduces an extra forced open algorithm, which is time-consuming. If the committer does not open the commitment, other parties can open the commitment by performing a predefined amount of computation. Two TC-based DRB protocols have been proposed [34, 35]. Both utilize a PBB to collect TCs from participants. The outputs are determined using these secret values within the TCs. The difference between these two approaches lies in their operation sequences. The protocol proposed in 2021 [34] combines TCs through homomorphic operations and then uses a forced open algorithm to obtain the output. In contrast, Bicorn published in 2023 [35] requires all nodes to reveal their secret values. If a party refuses, the forced open algorithm can still yield the final output.

As a summary, we compare the complexity of previous studies in Table 1 [7,17–22,24,27–29,32,33,35]. Notably as depicted in Table 1, threshold technique-based DRB protocols require high communication complexity, while existing TC-based DRB protocols require the usage of a PBB.

Both existing TC-based DRB protocols utilize a PBB to collect commitments from participants. The bulletin board ensures the immutability and public accessibility of messages and simplifies the process. Direct posting of messages eliminates the need for interactive communication among honest nodes. Utilizing a PBB results in a communication complexity of $O(n)$ per message. However, practical implementation often involves centralized services, such as Google's certificate transparency project or blockchain-based ledgers [36], which result in extra economic costs.

To achieve the functionality of a PBB, an alternative approach is to utilize a consensus protocol or broadcast channel. This approach has a communication complexity of $O(n^2)$ for a single message, as all nodes must communicate with each other to reach an agreement. However, directly utilizing a consensus protocol in a TC-based DRB protocol would result in a communication complexity of $O(n^2)$. This contradicts our goal of reducing the communication complexity of the protocol. Therefore, we need to design a new consensus protocol to meet our goal.

Besides, when a TC scheme is used in DRB protocols, it should satisfy an additional property. The evaluator of the forced open algorithm needs to convince a third party that the secret value is correct, without requiring the third party to run the forced open algorithm again. This property is defined in [37] as publicly verifiable. However, their definition and construction are complicated. In this paper, we simplify their definition of publicly verifiable timed commitment and propose a new construction PVTC.

This paper presents a novel DRB protocol PltcRB (abbreviated from pipeline timed commitment randomness beacon) based on our new PVTC without utilizing a PBB. Our protocol achieves the amortized

communication complexity of $O(n)$, which is optimal in distributed systems. Concretely, we utilize TCs with a large time parameter, to make it force-opened and used after several rounds. So we design a multiple rounds consensus protocol to amortize the total communication complexity into several rounds. Besides, no additional cost of communication complexity is required, even if the malicious node refuses to open its commitment. As a trade-off, the total computation complexity of each round is $O(n)$ when the number of parties n is larger than the security parameter κ.

Additionally, we establish an ideal model for DRB protocols based on TC and provide formal definitions for the properties that the DRB protocols should satisfy. We also give the security proofs for PItcRB. Finally, we implement a prototype of our protocol in GoLang and conduct a performance analysis under different configurations.

2 PVTC

TC scheme [31,34,37,38] is an extension of the traditional commitment scheme. It includes an additional forced open algorithm that permits anyone to open the commitment locally after a certain amount of time without assistance from the committer. A TC is considered publicly verifiable if the forced open algorithm is designed to produce an additional proof, enabling a third party to use this proof to verify the correctness of the output of the forced open algorithm.

Public verifiability is a necessary property when a TC scheme is applied in DRB protocols. If the committer is malicious, then the corresponding commitment will be opened by the forced open algorithm. In this scenario, the evaluator of the forced open algorithm must provide proof to convince a third party that the forced open algorithm was executed correctly, eliminating the need for the third party to re-run the forced open algorithm. The original definition of a TC does not require such proof and may cause a problem where a third party cannot verify the correctness of the opened value. The concept of public verifiability is also proposed in [37]. In their construction, the commitment algorithm outputs a proof π_{dec}, and the forced open algorithm outputs a proof π_{fdec}. These two proofs are used in different verification algorithms to verify the correctness of the secret value. In our definition, the TCs will only be forced open, thus we remove the proof generated in the commitment algorithm. We use only one verification algorithm to verify the correctness of the secret value revealed by the forced open algorithm. Our definition of PVTC is as follows.

Definition 1 (PVTC)**.** A PVTC scheme contains the following algorithms.

• $\mathsf{Setup}(1^\lambda) \to \mathsf{crs}$. The setup algorithm takes the security parameter 1^λ as input and outputs a common reference string crs.

• $\mathsf{Commit}(\mathsf{crs}, t, s) \to (c, \mathsf{aux}, \pi_{\text{com}})$. The inputs include the common reference string crs, the time parameter t, and the secret value s. The algorithm outputs the commitment string c, an auxiliary information aux used for evaluating ForcedOp, and a proof π_{com} employed to verify the validity of c and aux.

• $\mathsf{VerifyCom}(\mathsf{crs}, c, \mathsf{aux}, \pi_{\text{com}}) \to 0/1$. The commitment verification algorithm is used to determine whether the tuple published by the committer is correctly formatted. If VerifyCom outputs 1, the verifier accepts the commitment and is convinced that the secret value can be revealed by running the forced open algorithm. Otherwise, the verifier rejects it.

• $\mathsf{ForcedOp}(\mathsf{crs}, c, \mathsf{aux}) \to (s, \pi_{\text{dec}})/\perp$. The inputs consist of the common reference string crs, the commitment string c, and auxiliary information aux. The outputs include the secret value s and a proof π_{dec}. This algorithm must take at least time t.

• $\mathsf{VerifyDec}(\mathsf{crs}, c, \mathsf{aux}, s, \pi_{\text{dec}}) \to 0/1$. Anyone can use the decommitment verification algorithm to efficiently verify the correctness of the value s revealed by the forced open algorithm.

2.1 Property requirements

A PVTC scheme should satisfy the following properties:

• **Binding.** For a valid commitment tuple $(c, \mathsf{aux}, \pi_{\text{com}})$ generated by $\mathsf{Commit}(\mathsf{crs}, t, s)$, the commitment cannot be opened as $s' \neq s$.

• **Hiding.** For any probabilistic polynomial time (PPT) adversary $\mathcal{A}$ with running time $t' < t$, given the transcripts generated during the TC protocol as input, the probability that $\mathcal{A}$ can successfully distinguish the commitment string c from a random string R is negligible.

• **Commitment verifiability.** The validity of a commitment tuple $(c, \mathsf{aux}, \pi_{\mathrm{com}})$ can be checked by the commitment verification algorithm $\mathsf{VerifyCom}$.

• **Sequentiality.** Evaluators with parallel processors cannot execute the forced open algorithm $\mathsf{ForcedOp}$ significantly faster than evaluators with a single processor.

• **Public verifiability.** The evaluator of $\mathsf{ForcedOp}$ can convince a third party that the output s is correctly generated by producing a proof π_{dec}. A third party can run a decommitment verification algorithm $\mathsf{VerifyDec}$ to efficiently check the correctness of s.

2.2 Preliminaries

Before presenting our design, we first give definitions of two non-interactive proofs and two cryptographic primitives.

2.2.1 *Non-interactive proofs of sequential computation*

In the constructions of VDFs [30, 39, 40], the evaluator needs to generate a sequential proof $\pi(\mathbb{G}, x, y, t)$ to prove that the output value $y = x^{2^t}$ is exactly evaluated based on x by iterated squaring for t times.

An efficient construction of this proof is presented by Wesolowski in [40]. Specifically, the group $\mathbb{G} = \mathbb{Z}_N^*$, where $N = pq$ for two distinct primes p and q. For a security parameter λ, denote $\mathsf{Prime}(\lambda)$ as the set of first 2^λ primes. The non-interactive proof $\pi(\mathbb{G}, x, y, t)$ is generated as follows:

• The prover computes $\ell = \mathcal{H}_p(x, y, t)$ using a hash function $\mathcal{H}_p$ that maps an arbitrary string into a random prime in $\mathsf{Prime}(\lambda)$;

• The prover then computes the proof $\pi_t = \pi(\mathbb{G}, x, y, t) = x^{\lfloor 2^t/\ell \rfloor}$, and sends ℓ and the proof π_t to the verifier;

• The verifier checks $\ell = \mathcal{H}_p(x, y, t)$ and then computes $r = 2^t \bmod \ell$;

• The verifier accepts y if $\pi_t^\ell x^r = y$.

2.2.2 *Non-interactive zero-knowledge proofs of discrete logarithm knowledge*

A non-interactive zero-knowledge proof $\mathrm{EQLOG}(g, \hat{g}, h, \hat{h})$ is used to demonstrate the knowledge of a value α such that $\hat{g} = g^\alpha$ and $\hat{h} = h^\alpha$ without revealing any information about α when given a tuple $(g, \hat{g}, h, \hat{h})$. This proof is given by Chaum and Pedersen in [41] and the non-interactive version protocol works as follows:

• The prover selects a random value β and sends $g_1 = g^\beta$ and $h_1 = h^\beta$ to the verifier;

• The prover computes $e = \mathcal{H}_e(g, \hat{g}, h, \hat{h}, g_1, h_1)$ by using a hash function $\mathcal{H}_e$ and sends $z = \beta - \alpha e$ to the verifier;

• The verifier accepts if both $g_1 = g^z \hat{g}^e$ and $h_1 = h^z \hat{h}^e$ hold.

2.2.3 *TSs*

A (d, n)-TS scheme [42, 43] allows any d or more nodes of a group of n nodes to collaboratively generate a signature on a message. Specifically, a (d, n)-TS scheme TS, consists of the following algorithms: the setup algorithm $\mathsf{Setup}(1^\lambda)$, takes the security parameter 1^λ as input and produces a public key and partial secret keys for each of n nodes. Given a message m, node i uses its secret key to generate a partial signature $\sigma_i \leftarrow \mathsf{PartialSign}(m)$ on the message m. A set of partial signatures $\{\sigma_i\}_{i \in I}$ where I is a subset of n nodes and $|I| \geqslant d$, can be used to create a signature σ on the message m using the combine algorithm $\mathsf{Combine}(\{\sigma_i\}_{i \in I})$. σ can then be verified by the verification algorithm $\mathsf{Verify}(m, \sigma)$ using the public key. If an adversary possesses fewer than d partial signatures on message m, then the probability that it can construct a signature σ on the message that passes the verification algorithm is negligible.

2.2.4 *Verifiable random function*

A verifiable random function (VRF) [44] enables an evaluator to produce a publicly verifiable pseudorandom value. Specifically, a VRF comprises three key algorithms: the key generation algorithm $\mathsf{KGen}(1^\lambda)$, takes the security parameter 1^λ as input and generates a secret/public key pair $(\mathrm{sk}, \mathrm{pk})$ for the evaluator. When given an input x, the evaluator runs the evaluation function $\mathsf{Eval}(\mathrm{sk}, x)$ using its secret key to produce a value y and a proof π. A third party can verify the correctness of the output y by the verification

algorithm $\mathsf{Verify}(\mathrm{pk}, x, y, \pi)$. For any input x that has not been previously queried by the evaluator, the output y is pseudorandom.

2.3 Our construction

We give a new construction of PVTC scheme, which serves as an underlying subprotocol in the design of our DRB protocol PltcRB. The constructions of TCs in [31,45] utilize iterated squaring on RSA groups. They can be extended to any group of unknown order, for example, the class group of the imaginary field. Refs. [31,45] focus on the TC construction and do not consider the publicly verifiable problem. In our construction, we use an intermediate value computed during the forced open algorithm as the public proof of the correctness of the solution within the TC. The specific construction of PVTC is given as follows.

- $\mathsf{Setup}(1^\lambda, t) \to \mathsf{crs}$. The setup phase generates a group $\mathbb{G}$ such that no one knows its order. g is a random generator in $\mathbb{G}$, $h = g^{2^t}$ and $h' = g^{2^{t-1}}$ are two elements in $\mathbb{G}$ computed from g. A proof $\pi_t = \pi(\mathbb{G}, g, h, t)$ is used to efficiently verify that h equals g^{2^t}. Besides, a hash function $\mathcal{H} : \mathbb{G} \to \{0,1\}^{\ell(\lambda)}$ that maps an element in $\mathbb{G}$ to a pseudorandom string of length $\ell(\lambda)$-bits is generated. $\ell(\lambda)$ is the length of commitment string. The common reference string crs contains the group $\mathbb{G}$, the time parameter t, the elements g, h', h, the proof π_t and the function $\mathcal{H}$.
- $\mathsf{Commit}(\mathsf{crs}, s) \to (c, \mathsf{aux}, \pi_{\mathrm{com}})$. For a secret value $s \in \{0,1\}^{\ell(\lambda)}$, the committer p chooses a random value $b \in \mathbb{Z}_B$, where B is the upper bound of the order of $\mathbb{G}$. Then, p computes $\hat{g} = g^b, \hat{h} = h^b$ and $\hat{h'} = (h')^b$. The public auxiliary information is $\mathsf{aux} = (\hat{g}, \hat{h})$, while $\hat{h'}$ is kept secret and is used to encapsulate the secret value s. p then proves that $\log_g \hat{g} = \log_h \hat{h}$ by generating the proof $\mathrm{EQLOG}(g, \hat{g}, h, \hat{h})$. Finally, the proof $\pi_{\mathrm{com}} = \mathrm{EQLOG}(g, \hat{g}, h, \hat{h})$ and the commitment $c = \mathcal{H}(\hat{h'}) \oplus s$.
- $\mathsf{VerifyCom}(\mathsf{crs}, c, \mathsf{aux}, \pi_{\mathrm{com}}) \to 0/1$. Upon receiving the tuple $(c, \mathsf{aux}, \pi_{\mathrm{com}})$ from the committer, a receiver checks if $\hat{g}, \hat{h}$ are correctly computed from g, h based on π_{com}. If yes, then VerifyCom outputs 1, otherwise it outputs 0.
- $\mathsf{ForcedOp}(\mathsf{crs}, c, \mathsf{aux}) \to (s, \pi_{\mathrm{dec}})$. To forced open the TC c, one needs to parse $\mathsf{aux} = (\hat{g}, \hat{h})$ and compute $\hat{h'} = \hat{g}^{2^{t-1}} = (h')^b$ from $\hat{g}$ by iterated squaring for $t-1$ times. Then the secret value can be revealed by computing $s = c \oplus \mathcal{H}(\hat{h'})$. The proof π_{dec} is $\hat{h'}$.
- $\mathsf{VerifyDec}(\mathsf{crs}, c, \mathsf{aux}, s, \pi_{\mathrm{dec}}) \to 0/1$. If a third party receives the solution (s, π_{dec}) for the commitment c, it checks whether $\pi_{\mathrm{dec}}^2 = \hat{h}$ and $s = \mathcal{H}(\pi_{\mathrm{dec}}) \oplus c$ are both satisfied. If yes, then VerifyDec outputs 1, otherwise it outputs 0.

In our construction of PVTC, the time parameter t is defined in the setup algorithm, since all TCs used in our DRB protocol PltcRB rely on the same time parameter. Therefore, it can be incorporated into the setup algorithm as a component of the public parameter.

There are two distinct methods available for generating a group $\mathbb{G}$ of unknown order in the setup algorithm. The first approach utilizes the traditional RSA group. In this case, the RSA number N must be generated by multiple parties, ensuring that no individual node possesses the factorization of N. To achieve this goal, several research efforts have focused on generating RSA numbers in a distributed manner [46,47]. The second method employs the class group of an imaginary quadratic field [48,49]. By selecting a random large negative integer $d \equiv 1 \bmod 4$, the class group $\mathbb{G}$ of the imaginary quadratic field $\mathbb{Q}(\sqrt{d})$ exhibits a property: there exists no efficient algorithm to compute the order of $\mathbb{G}$ [48].

The proofs of binding, hiding, commitment verifiability, and sequentiality of our construction PVTC are similar to the proofs in [31,45]. The hiding of our construction additionally requires the zero-knowledge of the proof $\mathrm{EQLOG}(g, \hat{g}, h, \hat{h})$ that it does not reveal any information about the random value b. Besides, our TC satisfies public verifiability: for any evaluator of PVTC.ForcedOp, the proof π_{dec} can be obtained during the sequential computation. A third party can verify the revealed value by the proof π_{dec} using PVTC.VerifyDec.

3 PltcRB: a practical DRB protocol

This section proposes a specific construction of DRB protocol PltcRB that relies on PVTC. We begin by providing an overview of our protocol along with the models and assumptions of our construction.

3.1 Protocol overview

Our DRB protocol, PltcRB, is executed among a fixed set of $n = 3f + 1$ nodes, collectively denoted as $\mathcal{P} = \{p_1, \ldots, p_n\}$. These nodes are mutually distrusted, and at most f nodes can be corrupted by an adversary and deviate from the protocol arbitrarily. The remaining nodes are honest and strictly follow the protocol.

The protocol proceeds in rounds, producing a random number at the end of each round. In each round, a committer is selected at random from the set of committer candidates $\mathcal{V}$. Initially, this set includes all nodes in $\mathcal{P}$, but it may be changed as the protocol progresses. The committer generates and broadcasts a new TC to all nodes, with a long time parameter for the usage after n rounds. If a node accepts this TC, it responds to the committer with a partial signature on the TC. After receiving $2f + 1$ partial signatures from different nodes, the committer combines these partial signatures into a full signature by a $(2f + 1, n)$-TS scheme. The committer then broadcasts the full signature to all nodes. A TC is valid if there is a valid full signature on it.

However, a malicious committer might either fail to generate the full signature or only send it to a subset of honest nodes. All honest nodes should reach an agreement on the validity of TCs due to potential malicious behavior. Therefore, each round should have a consensus leader in addition to a committer. The consensus leader is designated to detect possible malicious behavior of the committer and to facilitate agreements among honest nodes on the TCs. As each TC is used after n rounds, honest nodes only need to reach an agreement before the TC is used. Therefore, multiple rounds can be employed to agree on the validity of one TC. If a committer is found to be malicious, it will be excluded from the set of committer candidates $\mathcal{V}$, and the corresponding TC will be marked as invalid and replaced with a default value.

After receiving a TC, honest nodes should execute the forced open algorithm. This prevents the malicious committer from refusing to open the commitment when it is used. The binding property of the TC ensures that only one value can be revealed as the secret value under the TC. To reduce computational complexity, a part of honest nodes rather than all honest nodes can be randomly selected to run the forced open algorithm.

The output of the beacon for each round is evaluated based on the secret value in the TC generated n rounds before. To achieve public verifiability, the protocol also produces a public proof for each round. This proof comprises the TC used in this round, the full signature on it, the corresponding secret value, and a forced open proof to verify the correctness of the secret value. If the TC is invalid, the output is evaluated based on a default value, and the public proof contains evidence of the malicious behavior of the committer.

3.2 Assumptions and models

3.2.1 *Computational power of nodes*

The sequentiality of a TC scheme ensures that no one can execute the forced-open algorithm significantly faster than others. Nevertheless, when a TC scheme is used in practice, it becomes imperative to consider the actual run time of the forced open algorithm. The run time is affected by the execution environment of the algorithm from several aspects, such as the optimization of the software implementation and the hardware capabilities. Consequently, disparities in the execution environment may lead to divergent run times, raising potential security concerns. Thus, it is crucial to describe the computational power of the nodes when implementing a randomness beacon protocol.

In this paper, we use a constant $\alpha \geqslant 1$ to represent the maximum disparity in computational power between the adversary and honest nodes. We assume that all honest nodes have the same computational power, while the computational capacity of the adversary might surpass that of honest nodes due to the use of superior hardware. Consequently, the difference in execution time of the forced open algorithm between honest nodes and the adversary can be constrained by α. We assume that α falls within a reasonable range when the protocol is implemented in practice.

3.2.2 *Adversary model*

The adversary always seeks to make the protocol deviate from meeting the security requirements. In the context of a DRB protocol, an adversary may attack the liveness or the consistency of the protocol, causing honest nodes to either fail to obtain the output or receive different outputs. Besides, an adversary

may seek to predict or bias the output of a randomness beacon, to obtain the output earlier than honest nodes or bias the outputs to the adversary's advantage.

The adversary should identify the corrupted nodes before the protocol execution, and it can control at most f nodes if there are total $n = 3f + 1$ nodes. The adversary has complete control over all corrupted nodes and knows all the messages transmitted and received by them. Furthermore, the adversary is a PPT adversary, which means it cannot break the underlying cryptographic primitives used in the protocol, including the signature scheme, the hash function, and other relevant components.

When the protocol is executed in practice, we assume that the adversary has more computational power than honest nodes. As mentioned earlier, we use a constant $\alpha \geqslant 1$ to limit the difference in computational power. In other words, if honest nodes can solve a TC in time t, then the adversary can solve the same TC in time t/α.

3.2.3 *Communication model*

PltcRB employs a synchronized network model with a reliable authenticated point-to-point message channel. The messages exchanged among honest nodes must be delivered within a certain time-bounded delay δ. Malicious nodes are unable to impede the transmission of messages between honest nodes. It is assumed that the communication delay is considerably small in comparison to the time parameter t of the TC scheme. In practice, the communication delays are typically on the order of milliseconds, while the time parameter of the TC usually ranges from seconds to minutes. Throughout the protocol, all messages sent should be properly signed, thereby ensuring that the receiver can ascertain the original sender of each message. Let $\langle m \rangle_{p_i}$ denote a message m signed by node p_i.

3.3 Protocol details

Now we present the details of our newly designed DRB protocol PltcRB.

3.3.1 *Setup phase*

At the beginning of the whole protocol, all nodes in $\mathcal{P} = \{p_1, \ldots, p_n\}$ initiate the execution of two distinct setup algorithms: the TC scheme $\mathsf{PVTC.Setup}(1^\lambda, n \cdot t)$ and the $(2f + 1, n)$-TS $\mathsf{TS.Setup}(1^\lambda)$. The first setup algorithm is employed to generate the common reference string crs with the security parameter denoted as 1^λ and the time parameter as $n \cdot t$, where t represents the time interval of a single round. The second setup algorithm uses the same security parameter 1^λ and generates the public key and partial signing keys for each node. The public key is publicly available, while each node maintains its partial signing key in secrecy.

After the public parameters are generated, the protocol starts at round 1. The initial phase of the protocol is from round 1 to round $2n$, during which no output is generated. In each of these initial rounds $1 \leqslant k \leqslant 2n$, one committer $p_{(k-1 \bmod n)+1}$ publishes a TC tc_{k+n} to all nodes. The specific publishing process is presented in Subsection 3.3.2. For $n + 1 \leqslant k \leqslant 2n$, the TC tc_k will be force opened in round k. Hence in round $2n$, a random initial value r can be evaluated based on these n TCs[1)].

3.3.2 *TC publish*

In round $k > 2n$, the committer c_k is publicly and randomly chosen from the set of committer candidates $\mathcal{V}$. For example, c_k is chosen based on the output of previous round: $c_k = \mathcal{V}[R_{k-1} \bmod |\mathcal{V}|]$, where $|\mathcal{V}|$ is the size of $\mathcal{V}$ and $\mathcal{V}[i]$ refers to the i-th element in $\mathcal{V}$ (with indexing starting from 0). Initially, $\mathcal{V} = \mathcal{P}$ comprises all participants.

Once the committer c_k is determined, it creates a TC tc_{k+n} to commit to its secret value r_{k+n} by using $\mathsf{PVTC.Commit}(\mathsf{crs}, r_{k+n})$. As indicated by the subscript, this TC will be used in round $k + n$. Subsequently, c_k sends tc_{k+n} to all nodes by broadcasting a proposal message $\langle \textsc{proposal}, tc_{k+n}, k \rangle_{c_k}$. Upon receiving a proposal message from the c_k, node p_i performs two checks. First, it ensures that it has not received another proposal message containing a different TC tc'_{k+n} from c_k. Second, it verifies the validity of the TC c_{k+n} using PVTC.VerifyCom. If either one of two checks is not passed, p_i will broadcast the invalid messages to all nodes as evidence that c_k is malicious. Then c_k will be excluded from the set of committer candidates $\mathcal{V}$. Otherwise, p_i accepts the proposal message.

1) If a faster bootstrapping phase is required, a multi-party protocol with higher communication complexity can be used.

If the proposal message passes two checks, p_i generates a partial signature $\sigma_{i,k+n}$ on the TC tc_{k+n} by the partial sign algorithm $\mathsf{TS.PartialSign}$ and sends an acknowledge message $\langle \text{ACKNOWLEDGE}, tc_{k+n}, \sigma_{i,k+n}, k\rangle_{p_i}$ back to c_k. Once c_k receives $2f+1$ valid acknowledge messages from different nodes, it can generate a full signature σ_{k+n} on its TC using the combination algorithm $\mathsf{TS.Combine}$. Subsequently, c_k broadcasts σ_{k+n} to all nodes. A node has received a valid TC tc_j if it has both the TC tc_j and the full signature σ_j.

3.3.3 *Agreement on TCs*

However, a malicious committer might send different messages to different honest nodes. So we need to make all honest nodes agree on the TC before it is used. At each round k, a consensus leader ℓ_k is uniquely chosen by a deterministic leader election algorithm. This algorithm should ensure that each node becomes the consensus leader once in any consecutive n rounds. For example, let $\mathcal{L}$ denote a random permutation of $\mathcal{P}$. The consensus leader of round k can be evaluated as $\ell_k = \mathcal{L}[(k \bmod n)+1]$. Here, $\mathcal{L}[i]$ refers to the i-th element in $\mathcal{L}$ (with indexing starting from 0). Since a TC will be used after n rounds, honest nodes only need to reach an agreement on a TC before it is used. To reduce communication complexity, nodes only communicate with the consensus leader in each round and multiple rounds are needed to reach an agreement on a TC.

Specifically, in round k, the leader ℓ_k checks whether it has received all valid TCs tc_i for $k < i \leqslant k+n$. If ℓ_k already possesses all valid TCs tc_i for $k < i \leqslant k+n$, it takes no further action. However, if ℓ_k lacks a valid TC tc_j, it will send a request message $\langle \text{REQUEST}, j, k\rangle_{\ell_k}$ about tc_j to all nodes, seeking to obtain the valid TC from other nodes. The leader is restricted to sending at most three request messages, and the request messages should be about the TC with the smallest subscripts. For example, if ℓ_k lacks $tc_{k+3}, tc_{k+6}, tc_{k+7}, tc_{k+9}$, it should send request messages about $tc_{k+3}, tc_{k+6}, tc_{k+7}$ to other nodes. However, ℓ_k does not generate a request message about tc_{k+9} and instead awaits information from other nodes about it.

Upon receiving the request message about TC tc_j from the leader, a non-leader node p_i responds based on its state at round $j-n$ (the round where all nodes expect to receive tc_j and σ_j). If p_i has received the valid TC tc_j at round $j-n$, then it sends a reply message $\langle \text{REPLY}, tc_j, \sigma_j, i, k\rangle_{p_i}$ to ℓ_k; Otherwise p_i replies a reply-empty message $\langle \text{REPLY-EMPTY}, j, i, k\rangle_{p_i}$ to the leader. If p_i receives more than three request messages from the consensus leader in the same round, then p_i sends all received request messages to all other nodes as evidence that the leader is malicious.

Note that the reply-empty messages imply potential malicious behavior of the committer. Once the leader ℓ_k receives $f+1$ or more valid reply-empty messages from different nodes for the same TC, it generates a blame message $\langle \text{BLAME}, \mathcal{E}, j, k\rangle_{\ell_k}$ containing all reply-empty messages it received and broadcasts the blame message to all nodes. Here $\mathcal{E}$ is the set of all reply-empty messages ℓ_k received at round k. The valid blame message indicates the malicious behavior of a committer, so it should be broadcast to all nodes promptly. When a node receives a valid blame message for the first time, it forwards the blame message to all nodes even if it is not the leader. This ensures all honest nodes receive the blame message in a short time: once an honest node has a blame message, then all honest nodes will receive it in time δ. Upon receiving a valid blame message for TC tc_j, the node replaces the value of tc_j with a default value $\perp$, and the associated committer of tc_j will be excluded from the set of committer candidates $\mathcal{V}$.

If the leader ℓ_k does not receive enough reply-empty messages, it must receive a valid TC tc_j from some honest nodes. Then ℓ_k sends tc_j to all non-leader nodes. If the leader both receives a valid TC and generates a blame message for the same TC, it adopts the blame message and discards the TC, as a valid blame message signifies the malicious behavior of the adversary.

3.3.4 *Forced open of TCs*

A malicious committer may refuse to open its commitment, so honest nodes need to run the forced open algorithm to ensure the timely opening of a TC. However, the time parameter of all TCs is set as $n \cdot t$, which is large. Requiring all honest nodes to run the forced open algorithm will incur high computation costs. Therefore, we require only some of the honest nodes to run the forced open algorithm by a VRF. Specifically, an honest node computes $y = \mathsf{VRF.Eval}(\text{sk}, k)$ and checks whether $y/2^{\ell(Y)} < v$ once it receives the TC tc_{k+n} at round k (even if it has not received the valid signature on tc_{k+n}). Here, $\ell(Y)$ denotes the length of VRF outputs and v represents the probability that one node is chosen. If the condition is met, this node is chosen to run the forced open algorithm of TC tc_{k+n}. Otherwise, it will do nothing

after receiving tc_{k+n}. Note that the computation of VRF requires the secret key of each node, so the adversary cannot predict which honest nodes have been selected.

If a TC tc_{k+n} is accepted by all honest nodes, then at least $f+1$ honest nodes have received this TC at round k. The expected minimum number of nodes to be chosen is $v\cdot(f+1)$. For example, if $v=1/(f+1)$, only one honest node is expected to be chosen for the forced open algorithm. However, the probability that all honest nodes fail to be chosen is $\Pr_{\text{fail}}=(1-v)^{f+1}$, lies within the range $(1/\mathrm{e}, 1/2]$ for all $f>0$, which is non-negligible. Here $\mathrm{e}\approx 2.718$ is a mathematical constant. To reduce the probability $\Pr_{\text{fail}}$, each node can compute VRF.Eval for κ different inputs For example, for $i=1,2,\ldots,\kappa$, the node computes $y_i=\mathsf{VRF.Eval}(\mathrm{sk}, k||i)$. The node would be selected if any of the computed y_i satisfies the condition $y_i/2^{\ell(Y)}<v$. Here κ is a parameter that effects the value of $\Pr_{\text{fail}}$. In this method, the upper bound of $\Pr_{\text{fail}}$ is $1/2^{\kappa}$ and it becomes negligible when κ is large enough. In other words, the probability that at least one honest node is chosen is overwhelming. Since honest nodes will strictly follow the protocol, as long as one honest node is chosen to run the forced open algorithm, the TC can be revealed when it is used.

3.3.5 *Output phase*

After an honest node finishes the evaluation of the forced open algorithm, it needs to broadcast the result, along with the output of VRF indicating its selection for evaluation, to all other nodes. This ensures that all honest nodes have the result when a TC is used.

The output of round k utilizes the secret value r_k from TC tc_k. The evaluation can be represented as $R_k=\mathcal{H}(k, R_{k-1}, r_k)$, where R_{k-1} is the output of previous round. As part of the output phase, a proof π_k is generated, consisting of the TC tc_k, the secret value r_k, the corresponding proof π_{dec} and the signature σ_k. The binding property guarantees the uniqueness of the value r_k when given a TC tc_k. So a third party can efficiently verify the output through PVTC.VerifyDec. In the case that a blame message about the committer of tc_k is generated, the output R_k is computed based on a default value $\perp$, and the proof π_k contains the blame message about the committer.

4 Analysis of PltcRB

This section represents the security and complexity analysis of PltcRB.

4.1 Security analysis of PltcRB

4.1.1 *Agreement on the TCs*

First, we need to prove that honest nodes can reach an agreement on the validity of a TC before it is used.

Lemma 1. In round k, the committer c_k can generate at most one valid TC.

Proof. In round k, a valid signature σ_k for a proposal message requires at least $2f+1$ partial signature on it, which indicates that at least $f+1$ honest nodes agree on it. If there is more than one valid signature σ_k and σ_k' for different proposal messages in one round, there must be two sets of $f+1$ honest nodes that provide their partial signatures. Since there are only $2f+1$ honest nodes, according to the pigeon nest principle, there must be one honest node that generates two valid partial signatures on different proposal messages, which contradicts the assumption of honest nodes. So in one round, at most one valid signature on the proposal message can be generated.

Theorem 1 (Agreement on the validity of TCs)**.** For each TC tc_j, all honest nodes reach an agreement on its validity before round j.

Proof. If the TC tc_j is generated by an honest committer, it is obvious that all honest nodes agree on its validity.

If a TC tc_j is generated by a malicious committer, then each honest node is in one of the following three states when it becomes the consensus leader: (1) it has received a valid tc_j; (2) it has not received a valid tc_j but can generate a request message about tc_j; (3) it has not received a valid tc_j and is unable to generate a request message. According to protocol, if an honest consensus leader sends a request message about TC tc_j, all honest nodes agree on its validity. Therefore, the rest honest nodes will not generate a request message about tc_j again.

Since there are at most $n-1=3f$ TCs that have smaller subscripts than tc_j, so at most f honest nodes are in state (3) when it becomes the consensus leader before tc_j is used. In the worst case, these f honest nodes cannot receive a valid tc_j before round j. However, they are convinced that at least $f+1$ honest nodes have received valid tc_j, otherwise, there would be a blame message about tc_j before round j. Hence, they believe that the TC tc_j is valid.

In summary, before a TC tc_j is used, all honest nodes will agree on the validity of tc_j.

4.1.2 *Liveness*

PltcRB is expected to output a random number at each round. It uses a VRF to choose some honest nodes to run the forced open algorithm. The liveness of the protocol requires that at least one honest node runs the forced open algorithm.

Lemma 2. For each TC tc_k, if it is used to compute the output for round k, then the probability that no honest node is chosen to compute the forced open algorithm is negligible.

Proof. If a TC tc_k is generated by the honest committer, all $2f+1$ honest nodes should have received tc_k at round $k-n$. However, if tc_k is generated by a malicious committer and there is a valid signature about tc_k, then at least $f+1$ honest nodes should have received tc_k at round $k-n$. So for a valid TC, the number of honest nodes that received a TC in time is at least $f+1$.

According to the protocol, each honest node that receives a TC at the corresponding round computes a VRF for λ different inputs, then the probability that none of these λ outputs is chosen is $(1-v)^{\lambda}$, where $v=1/(f+1)$ is the probability that one node is chosen. Hence the probability that no honest node is chosen is at most $(1-v)^{\lambda\cdot(f+1)}<\frac{1}{2^{\lambda}}$.

So if a TC is finally used, there is an overwhelming probability that at least one honest node is chosen to run the forced open algorithm of TC.

Theorem 2 (Liveness)**.** At the end of each round k, PltcRB can output a random number R_k with overwhelming probability.

Proof. Theorem 1 ensures that at the beginning of round k, all honest nodes reach agreement on the validity of tc_k. If tc_k is accepted by all honest nodes, Lemma 2 ensures that the probability of no honest node computing the forced open algorithm is negligible. So the value within TC tc_k can be revealed and R_k can be then evaluated with overwhelming probability. In the case that the TC tc_k is a malicious one, the blame message mechanism ensures that all honest nodes can receive the blame message, and the output R_k is evaluated by a default value.

Hence at the end of a round k, the protocol can output a random number R_k with overwhelming probability.

Theorem 3 (Output uniqueness)**.** For each round k, there is only one valid beacon output R_k.

Proof. The binding property of TC ensures that a TC can only be opened to a unique value. Lemma 1 ensures that all honest nodes are in the same state about the TC tc_k. So a third party can obtain the state of TC by receiving the same state from $f+1$ different participants, and the output R_k is uniquely determined by the value r_k under the TC tc_k or a default value $\perp$.

4.1.3 *Unpredictability*

Unpredictability captures the ability of an adversary to predict the outputs of future rounds. In our protocol, the unpredictability is related to the constant α that describes the computational power difference between the adversary and the honest nodes.

Theorem 4 (Unpredictability)**.** Assuming the computation power of the adversary is α times better than the honest node, the adversary can predict the outputs of $n\cdot(1-1/\alpha)$ rounds later, and the probability for the adversary to predict the output of future rounds decreases exponentially as the round number increases.

Proof. For an adversary wants to predict the output R_{k+d} before the end of round k, it should know both the value behind the TC tc_{k+d} and the output R_{k+d-1} before the end of round k.

An adversary can obtain the advantage of predicting future outputs from two aspects. One is that the computation power of the adversary is higher than honest nodes, hence the adversary can solve the TC of honest nodes before n rounds. Another one is that the secret values in TCs of corrupted nodes are known to the adversary.

According to the assumption, if the computation power of the adversary is α times better than honest nodes, then the time that is needed to solve a TC of honest node is $\lceil n/\alpha \rceil$ rounds. For a TC tc_{k+d}, if the time that the adversary has received the TC is longer than the time for the adversary to force open the TC, then it can reveal the value in this TC at round k. Hence we have $n - d \geqslant \lceil n/\alpha \rceil \geqslant n/\alpha$, which indicates that $d \leqslant n \cdot (1 - 1/\alpha)$.

The adversary can also obtain the advantage of predicting the outputs of future rounds by being chosen as the committers for continuous rounds. For example, if the committer of tc_{k+d+1} is controlled by the adversary, then the adversary can predict the output of the corresponding round, However, the probability that the committers of continuous rounds are controlled by the adversary will decrease exponentially as the round number increases.

So at the beginning of round k, if the computation power of the adversary is α times better than the honest node, then it can predict the output of round $k + n \cdot (1 - 1/\alpha)$. The probability that the adversary can predict the outputs of future rounds after round $k + n \cdot (1 - 1/\alpha)$ will decrease exponentially as the round number increases.

4.1.4 *Bias-resistance*

A DRB protocol is bias-resistance if no PPT adversary can bias the output with non-negligible probability.

Theorem 5 (Bias-resistance)**.** During the execution of PltcRB, the probability that a PPT adversary can bias the output R_k of round k is negligible.

Proof. In PltcRB, the output of round k is based on a TC published at round $k - n$ and the output of round $k - 1$. If an adversary can bias the output R_k, then it must satisfy one of the following two conditions: (1) it can force open the TC tc_k and can bias the output R_{k-1} of round $k - 1$; (2) it is the committer c_{k-n} and can predict the output R_{k-1}.

The first case is impossible since the output R_{k-1} is determined before the TC tc_k is published, even if the adversary knows the value of tc_k, the output R_{k-1} is already determined.

In the second case, the adversary is the committer c_{k-n}. It can generate a TC with its secret value, or it can generate an invalid TC to use the default value. Besides, the adversary needs to predict the output of round $k - 1$ when it is chosen as the committer in round $k - n$. According to Theorem 4, it means that the adversary has much higher computational power than honest nodes, i.e., $\alpha > n$, which is not reasonable in practice; or the adversary is chosen to be the committers of consecutive rounds before tc_{k-1}, which is negligible.

So during the execution of PltcRB, the probability that an adversary can bias the output is negligible.

Discussion about α. The difference in computational power between honest nodes and malicious adversaries always exists. According to the performance data given by Intel [50], the difference between updated server-grade and updated custom-grade chips is approximately 10. Hardware performance also doubles roughly every four years, as indicated by the performance growth rate given in [51].

Therefore, if adversaries use updated server-grade chips while honest nodes employ four years old custom-grade chips, the performance gap widens to approximately 20 times. Consequently, when implementing our protocol in practice, we require honest nodes to periodically upgrade their hardware to make sure that the disparity α remains within a reasonable range. This represents a tradeoff between the security and the hardware requirements of honest nodes. This is an inherent issue when time-related primitives are used in practice. As discussed in [32], adversaries can always gain a time advantage if they possess more computational power than honest nodes.

4.1.5 *Public verifiability*

Theorem 6 (Public verifiability)**.** At the end of each round k, given the output R_k and the proof π_k, a third party can verify the correctness of R_k.

Proof. After a third party receives the output R_k and the proof π_k of round k, it first checks the validity of TC tc_k by $f + 1$ same states from different nodes. If the TC tc_k is valid, the proof π_k contains the TC tc_k, the valid signature σ_k, and the secret value r_k. The verifier first checks whether σ_k is the valid signature on the proposal message and whether r_k is the correct value under the TC tc_k. Finally, it checks that the output R_k is correctly computed using r_k.

Otherwise, if the TC tc_k is invalid, the proof π_k contains the blame message about the committer. Then a third party checks whether the output R_k is correctly computed based on the default value $\perp$.

4.2 Complexity analysis

Now we provide a theoretical perspective on the complexity analysis of our DRB protocol PltcRB.

The setup phase requires communication between all nodes to generate public parameters, such as the group $\mathbb{G}$ and the public key of the TS scheme. The communication complexity of the setup phase is $O(n^2)$, while the computation complexity is $O(n)$. Besides, PltcRB requires $n+1$ random numbers of the initial n rounds, which brings $O(n^3)$ communication complexity.

The communication complexity for the TC publish phase is $O(n)$ where the committer sends TCs to other nodes, collects partial signatures, and broadcasts the full signature. The malicious behavior during this phase will be detected, so it prevents the increase in complexity. In the agreement phase, nodes only interact with the consensus leader of the current round. Each node replies to at most three request messages from the consensus leader, so a malicious leader cannot increase the communication complexity by sending many request messages to all nodes. Hence the communication complexity of the agreement phase is $O(n)$.

According to the protocol, each honest node repeats the VRF computation κ times upon receiving a TC. The probability of being selected each time is $v = 1/(f+1)$. So the expected number of honest nodes to run the forced open algorithm for one TC is $O(\kappa)$ when the number of participants n is larger than the security parameter κ. The output phase requires the nodes selected for the forced open algorithm to broadcast results to all nodes, incurring a communication complexity of $O(\kappa n)$. The malicious nodes cannot increase the complexity of the output phase since the output of VRF is required in the output phase.

The computation complexity is related to the number of honest nodes that compute the forced open algorithm. Hence the computation complexity of PltcRB is $O(\kappa n)$ when n is larger than the security parameter κ.

Note that the broadcast of the blame message brings additional high communication costs. However, once a valid blame message is generated, the corresponding committer will be excluded from the set of committer candidates. So it will only happen for at most f times during the whole execution of PltcRB and it will not affect the amortized complexity of the protocol.

In summary, the amortized communication complexity for generating a random number is $O(n)$ and the computation complexity is $O(n)$ for a large number of participants n.

5 Experiment and evaluation

This section presents a comprehensive evaluation of the performance of PltcRB through experiments. We start by explaining our implementation and then dive into a detailed analysis of the experimental results. Our evaluation focuses on two key metrics: the efficiency of generating, verifying, and forced-opening TCs, which directly effects the time performance of PltcRB, and the bandwidth cost, which influences the scalability and usability of PltcRB.

5.1 Experiment setup

We have implemented PltcRB using Go, and it is available on GitHub[2)]. We select the class group as the group of unknown order in our proof-of-concept implementation. To support calculations on class groups and advanced cryptographic primitives, we employ the libraries Alice and Kyber. Additionally, to ensure reliable communication, we utilize the gRPC library. Moreover, we use standard Go libraries for fundamental cryptographic and arithmetic operations.

The experiments are conducted on a laptop with an Intel Core i5-10210U (1.60 GHz) processor, 16 GB of RAM, and running Ubuntu 22.04 LTS. To simulate a distributed system, each running node actively listens on specific ports for communication. Dedicated communication channels are created between connected nodes to enable efficient data exchange.

5.2 Analysis of temporal performance

The time interval in PltcRB is affected by the time parameter t and the security parameter λ. The time parameter t affects the running time of the forced open algorithm while the security parameter λ affects

2) https://github.com/Academic-Demos/Randomness-Beacon.

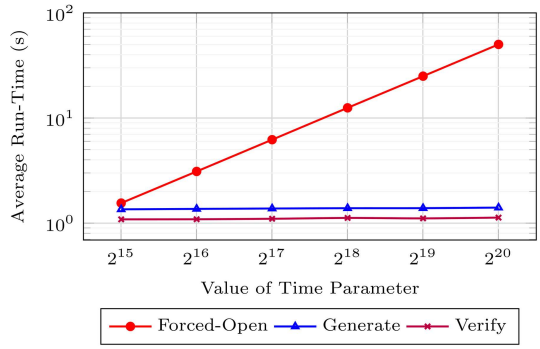

Figure 1 (Color online) Average run-time under different time parameters.

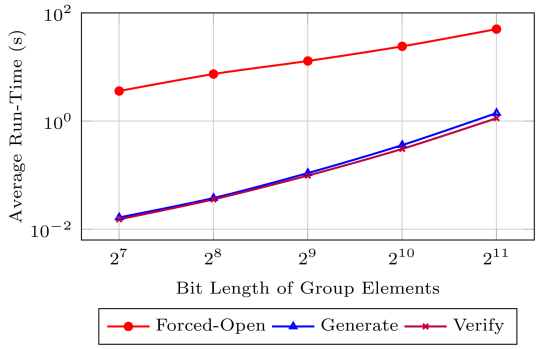

Figure 2 (Color online) Average run-time under different bit lengths.

the bit length of group elements. So it is essential to assess how these parameters influence the efficiency of generating, verifying, and forced opening TCs. This analysis will also contribute to evaluating the temporal performance of PItcRB.

First, we investigate how the efficiency of TC relates to the time parameter t by conducting experiments with t ranging from 2^{15} to 2^{20} while maintaining the length of the group element at 2048 bits. Then, we explore the correlation between TC efficiency and the bit length of group elements, by varying the bit length from 2^7 to 2^{11} and keeping the time parameter t fixed at 2^{20}. All other conditions are the same in these two experiments.

The results, as depicted in Figures 1 and 2 using logarithmic coordinates, indicate linear correlations between the average running time to force open a TC and the time parameter t and the bit length of group elements, respectively. The first correlation comes from an increase in the number of repeated squaring calculations as t increases. The second correlation comes from the increase in the time required for executing a single squaring calculation as the bit length of group elements grows.

Figures 1 and 2 also show that the time needed to generate and verify a TC is much smaller compared to the time needed to force open a TC. This indicates the efficiency of the committer and the verifier.

Consequently, the temporal performance of PItcRB is primarily determined by how efficiently the TC can be forced open. In practical implementations of a DRB protocol, the time interval between two rounds is set as 30 s or 1 min [16, 22]. So we set the time interval between the two rounds as 1 min. In this setting, the communication delay δ can be ignored since it is usually at the grade of milliseconds.

5.3 Comparison of performance

In this subsection, we compare the performance of our construction PItcRB to HeadStart [33], Hydrand [19], and Spurt [28]. The performance comparison is made from two aspects: the bandwidth usage and the throughput. The bandwidth usage represents the number of bytes sent and received for each beacon output across all nodes, and the throughput describes the number of outputs of the DRB protocol within a certain amount of time. Both Hydrand and Spurt are based on the PVSS scheme, while HeadStart is based on the VDF scheme. The communication complexities are $O(n^2)$ for Hydrand and Spurt, and $O(n \log n)$ for HeadStart. Our protocol has an optimal communication complexity of $O(n)$.

Since Spurt and Hydrand utilize the BLS12-381 curve, which achieves a 128-bit security strength, we choose a security parameter of 6656 bits for PItcRB, matching HeadStart's security level, as both achieve the same security strength as stated in [52]. Additionally, to assess scalability, we experiment with different node counts and compare the outcomes with the three other protocols.

As depicted in Figure 3, PItcRB uses the least bandwidth utilization compared to other protocols. Furthermore, the overhead in bandwidth usage increases approximately linearly with the number of nodes n. This linearity is a result of PItcRB achieving optimal communication complexity of $O(n)$. The relationship between bandwidth usage and the number of nodes n remains linear when keeping the class group constant.

The comparison of the throughput between our protocol and other protocols is shown in Figure 4. When the number of nodes is small, the throughput of our protocol is lower than that of other protocols. The declining trend indicates that the throughput of our protocol decreases at the slowest rate as the

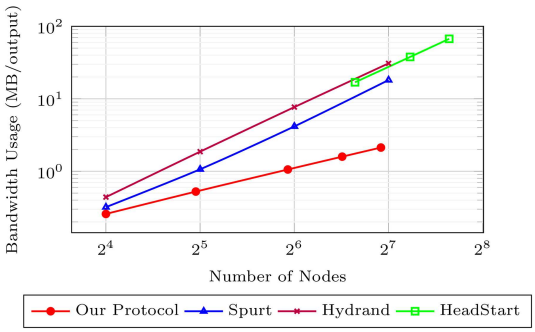

Figure 3 (Color online) Bandwidth usage under different numbers of nodes.

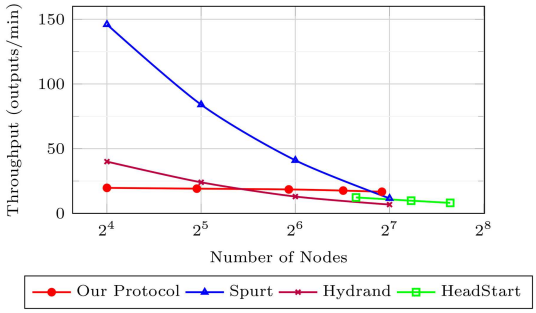

Figure 4 (Color online) Throughput under different numbers of nodes.

number of nodes increases.

6 Conclusion

In this paper, we present a novel construction of DRB protocol PltcRB based on publicly verifiable timed commitment and consensus protocol with multiple rounds. PltcRB achieves optimal amortized communication complexity while eliminating the requirement of a PBB. We establish that PltcRB successfully adheres to the essential properties of a DRB protocol, delving into the implications of the computational power discrepancy between adversarial entities and honest nodes on the protocol's behavior. Furthermore, we analyze the performance of PltcRB from both theoretical and experimental perspectives to demonstrate its efficiency. Although the complexity of the setup phase is high, it only executes once. Notably, the communication complexity and computation complexity are both $O(n)$ when n is larger than the security parameter κ. Subsequently, we implement PltcRB using GoLang and analyze its practical performance through experimental results. According to our experiments, the average running time to force open a TC exhibits a linear relationship with the time parameter t and the bit length of security parameter κ. Besides, PltcRB demonstrates bandwidth usage comparable to related protocols.

Acknowledgements This work was supported by National Key Research and Development of China (Grant No. 2019YFB2101600).

References

1 Mittal P, Olumofin F, Troncoso C, et al. PIR-Tor: scalable anonymous communication using private information retrieval. In: Proceedings of the 20th USENIX Security Symposium (USENIX security 11), 2011
2 van den Hooff J, Lazar D, Zaharia M, et al. Vuvuzela: scalable private messaging resistant to traffic analysis. In: Proceedings of the 25th Symposium on Operating Systems Principles, 2015. 137–152
3 Wolinsky D I, Corrigan-Gibbs H, Ford B, et al. Dissent in numbers: making strong anonymity scale. In: Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12), 2012. 179–182
4 Bonneau J, Clark J, Goldfeder S. On Bitcoin as a public randomness source. 2015. https://jbonneau.com/doc/BGC14-beacon-draft.pdf
5 Adida B. Helios: web-based open-audit voting. In: Proceedings of the USENIX Security Symposium, 2008. 335–348
6 Baigneres T, Delerablée C, Finiasz M, et al. Trap me if you can — million dollar curve. 2015. https://eprint.iacr.org/2015/1249.pdf
7 Lenstra A K, Wesolowski B. Trustworthy public randomness with sloth, unicorn, and trx. Int J Appl Crypto, 2017, 3: 330
8 Kiayias A, Russell A, David B, et al. Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Proceedings of the Annual International Cryptology Conference, 2017. 357–388
9 Gaži P, Kiayias A, Zindros D. Proof-of-stake sidechains. In: Proceedings of the IEEE Symposium on Security and Privacy (SP), 2019. 139–156
10 Kokoris-Kogias E, Jovanovic P, Gasser L, et al. OmniLedger: a secure, scale-out, decentralized ledger via sharding. In: Proceedings of the IEEE Symposium on Security and Privacy (SP), 2018. 583–598
11 Luu L, Narayanan V, Zheng C, et al. A secure sharding protocol for open blockchains. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2016. 17–30
12 Zamani M, Movahedi M, Raykova M. RapidChain: scaling blockchain via full sharding. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2018. 931–948
13 Zou W, Lo D, Kochhar P S, et al. Smart contract development: challenges and opportunities. IEEE Trans Softw Eng, 2021, 47: 2084–2106
14 Kalodner H, Goldfeder S, Chen X, et al. Arbitrum: scalable, private smart contracts. In: Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), 2018. 1353–1370
15 Rabin M O. Transaction protection by beacons. J Comput Syst Sci, 1983, 27: 256–267
16 Kelsey J, Brandão L T, Peralta R, et al. A Reference for Randomness Beacons: Format and Protocol Version 2. Technical Report NISTIR 8213, 2019
17 Syta E, Jovanovic P, Kogias E K, et al. Scalable bias-resistant distributed randomness. In: Proceedings of the IEEE Symposium on Security and Privacy (SP), 2017. 444–460

18 Cascudo I, David B. SCRAPE: scalable randomness attested by public entities. In: Proceedings of the International Conference on Applied Cryptography and Network Security, 2017. 537–556
19 Schindler P, Judmayer A, Stifter N, et al. HydRand: efficient continuous distributed randomness. In: Proceedings of the IEEE Symposium on Security and Privacy (SP), 2020. 73–89
20 Cachin C, Kursawe K, Shoup V. Random oracles in constantinople: practical asynchronous byzantine agreement using cryptography. J Crypto, 2005, 18: 219–246
21 Hanke T, Movahedi M, Williams D. Dfinity technology overview series, consensus system. 2018. ArXiv:1805.04548
22 DRAND. Drand: a distributed randomness beacon daemon, 2022. https://github.com/drand/drand
23 Galindo D, Liu J, Ordean M, et al. Fully distributed verifiable random functions and their application to decentralised random beacons. In: Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P), 2021. 88–102
24 Cherniaeva A, Shirobokov I, Shlomovits O. Homomorphic encryption random beacon. 2019. https://eprint.iacr.org/2019/1320.pdf
25 Nguyen-Van T, Nguyen-Anh T, Le T D, et al. Scalable distributed random number generation based on homomorphic encryption. In: Proceedings of the IEEE International Conference on Blockchain (Blockchain), 2019. 572–579
26 Cascudo I, David B. Albatross: publicly attestable batched randomness based on secret sharing. In: Proceedings of the 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, 2020. 311–341
27 Bhat A, Shrestha N, Luo Z, et al. Randpiper — reconfiguration-friendly random beacons with quadratic communication. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2021. 3502–3524
28 Das S, Krishnan V, Isaac I M, et al. SPURT: scalable distributed randomness beacon with transparent setup. In: Proceedings of the IEEE Symposium on Security and Privacy (SP), 2022. 2502–2517
29 Bhat A, Shrestha N, Kate A, et al. OptRand: optimistically responsive reconfigurable distributed randomness. 2023. https://eprint.iacr.org/2022/193.pdf
30 Boneh D, Bonneau J, Bünz B, et al. Verifiable delay functions. In: Proceedings of the Annual International Cryptology Conference, 2018. 757–788
31 Boneh D, Naor M. Timed commitments. In: Proceedings of the Annual International Cryptology Conference, 2000. 236–254
32 Schindler P, Judmayer A, Hittmeir M, et al. RandRunner: distributed randomness from trapdoor VDFs with strong uniqueness. 2020. https://eprint.iacr.org/2020/942.pdf
33 Lee H, Hsu Y, Wang J J, et al. HeadStart: efficiently verifiable and low-latency participatory randomness generation at scale. In: Proceedings of the Network and Distributed System Security (NDSS) Symposium, 2022
34 Thyagarajan S A K, Castagnos G, Laguillaumie F, et al. Efficient CCA timed commitments in class groups. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2021. 2663–2684
35 Choi K, Arun A, Tyagi N, et al. Bicorn: an optimistically efficient distributed randomness beacon. 2023. https://eprint.iacr.org/2023/221.pdf
36 Choudhuri A R, Green M, Jain A, et al. Fairness in an unfair world: fair multiparty computation from public bulletin boards. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security, 2017. 719–728
37 Chvojka P, Jager T. Simple, fast, efficient, and tightly-secure non-malleable non-interactive timed commitments. In: Proceedings of the IACR International Conference on Public-Key Cryptography, 2023. 500–529
38 Katz J, Loss J, Xu J. On the security of time-lock puzzles and timed commitments. In: Proceedings of the 18th International Conference on Theory of Cryptography, Durham, 2020. 390–413
39 Pietrzak K. Simple verifiable delay functions. In: Proceedings of the 10th Innovations in Theoretical Computer Science Conference, 2019
40 Wesolowski B. Efficient verifiable delay functions. In: Proceedings of the 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, 2019. 379–407
41 Chaum D, Pedersen T P. Transferred cash grows in size. In: Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques, 1992. 390–407
42 Shoup V. Practical threshold signatures. In: Proceedings of International Conference on the Theory and Application of Cryptographic Techniques, Bruges, 2000. 207–220
43 Boneh D, Lynn B, Shacham H. Short signatures from the weil pairing. In: Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security, 2001. 514–532
44 Micali S, Rabin M, Vadhan S. Verifiable random functions. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999. 120–130
45 Garay J A, Jakobsson M. Timed release of standard digital signatures. In: Proceedings of the International Conference on Financial Cryptography, 2002. 168–182
46 Gilboa N. Two party RSA key generation. In: Proceedings of the Annual International Cryptology Conference, 1999. 116–129
47 Boneh D, Franklin M. Efficient generation of shared RSA keys. In: Proceedings of the 17th Annual International Cryptology Conference, Santa Barbara, 1997. 425–439
48 Buchmann J, Hamdy S. A survey on IQ cryptography. In: Proceedings of the Public-Key Cryptography and Computational Number Theory, 2001. 1–15
49 Hafner J L, McCurley K S. A rigorous subexponential algorithm for computation of class groups. J Amer Math Soc, 1989, 2: 837–850
50 Intel. Export compliance metrics for intel microprocessors, 2022. https://www.intel.com/content/www/us/en/support/articles/000005755/processors.html
51 Hennessy J L, Patterson D A. Computer Architecture: A Quantitative Approach. San Francisco: Morgan Kaufmann Publishers Inc., 2011
52 Dobson S, Galbraith S D, Smith B. Trustless groups of unknown order with hyperelliptic curves. 2020. https://eprint.iacr.org/2020/196.pdf