Fair Delivery of Decentralised Randomness Beacon

Runchao Han[1,2] and Jiangshan Yu[1]

[1] Monash University
[2] CSIRO-Data61
me@runchao.rocks, jiangshan.yu@monash.edu

Abstract. The security of many protocols such as voting and blockchains relies on a secure source of randomness. Decentralised Randomness Beacon (DRB) has been considered as a promising approach, where a set of participants jointly generates a sequence of random outputs. While the DRBs have been extensively studied, they failed to capture the advantage that some participants learn random outputs earlier than other participants. In time-sensitive protocols whose execution depends on the randomness from a DRB, such an advantage allows the adversary to behave adaptively according to random outputs, compromising the fairness and/or security in these protocols.

In this paper, we formalise a new property, *delivery-fairness*, to quantify the advantage. In particular, we distinguish two aspects of delivery-fairness, namely *length-advantage*, i.e., how many random outputs an adversary can learn earlier than correct participants, and *time-advantage*, i.e., how much time an adversary can learn a given random output earlier than correct participants. In addition, we prove the lower bound of delivery-fairness showing optimal guarantee. We further analyse the delivery-fairness guarantee of state-of-the-art DRBs and discuss insights, which, we show through case studies, could help improve delivery-fairness of existing systems to its optimal.

1 Introduction

Decentralised Randomness Beacon (DRB) is a protocol where a set of participants jointly generates a sequence of random outputs. It has been a promising approach to provide secure randomness to other protocols and applications. There have been emerging DRB proposals [6, 22, 29, 41] and deployed DRB systems [3, 5], and DRBs have been used by many high-financial-stake applications such as blockchains [21, 30, 33, 35], lotteries [11], games [14, 17], and non-fungible tokens (NFTs) [1, 4].

Applications have two common approaches to use a DRB, namely 1) by using a random output at a certain height which the DRB has not reached yet, and 2) by using a random output produced near a certain time in the future. For example, Polygon Hermez [13] and Celo [2] used the 697500-th random output [10] and the random output produced near 29/10/2021 9am UTC [12] of Drand [6] for their zkSNARK trusted setup, respectively.

Existing DRBs are designed with three main security properties in consideration, namely *consistency*, *liveness* and *unpredictability* [22, 29]. Consistency states that all correct participants (who generate random outputs) share the same view on a unique ledger, i.e., sequence of random outputs. Liveness states that all correct participants produce random outputs no slower than a certain rate. Unpredictability states that no participant can distinguish a future random output from a uniformly sampled random string of the same length.

However, existing adversary models and security properties do not cover the unfair case resulted by the difference in the timing of learning a random output. In particular, when a random output is generated, the first "creator", or "observer", learns its value earlier than others. Such an advantage is not desired in practice. In time-sensitive protocols whose execution depends on the randomness from a DRB, the advantage allows an adversary to behave adaptively according to random outputs, compromising the fairness and/or security in these protocols. In the above example of zkSNARK trusted setup, if the adversary learns the random output before the trusted setup starts, then Hermez's and Celo's trusted setup will be insecure against an adaptive adversary [10,15,25]. Another example is the on-chain lottery which determines the winner out of all players by using random outputs from a DRB. If the adversary learns the random output before the lottery starts, then it learns whether it will win the lottery in advance, and thus can decide whether to participate in the lottery according to its outcome.

1.1 Related works

A systematic and formal study on the advantage of learning random outputs faster in DRB is still missing. Existing DRB models only focus on certain attacks leading to certain aspects in this advantage [22, 30]. Related security properties in other primitives do not cover this advantage, as they either concern eventual delivery without quantifying the advantage [27], or concern the advantage among correct participants excluding Byzantine participants [28, 37–39].

Related properties in DRBs. Previous research either informally studies such advantage, or formally studies certain attacks leading to this advantage. Ouroboros Praos [30] states that a DRB is leaky if the leader can learn the random output in the next epoch. However, the "leaky" definition is embedded in the ideal functionality of DRB rather than stated separately, disallowing specific analysis.

RandPiper [22] combines the "leaky" property into the unpredictability property, yielding d-unpredictability. It states that in the beginning of an epoch, the adversary can learn at most d future random outputs in advance. However, d-unpredictability only captures *length-advantage*, i.e., how many random outputs the adversary can learn earlier than correct participants, but not *time-advantage*, i.e., how much time the adversary can learn a given random output earlier than correct participants. In addition, RandPiper only studies d-unpredictability under the private beacon attack where the adversary solely samples random outputs, neglecting other possible attacks on delivery-fairness.

SPURT [29] defines the "nearly simultaneous beacon output" as a part of the unpredictability property. It states that all correct participants learn a random output within a constant time after the adversary learns it. The "nearly simultaneous" notion only captures the time-advantage but not the length-advantage. It also falls short of quantifying the time-advantage only asymptotically, and thus does not support concrete analysis.

Guaranteed output delivery and fairness in Multiparty Computation. Guaranteed output delivery (GOD) and fairness are properties of multiparty computation (MPC) protocols, where participants jointly compute a function of their inputs securely under a subset of corrupted participants. GOD specifies that corrupted participants cannot prevent the correct participants from receiving the function's output. Fairness specifies that corrupted participants should receive the function's output if and only if correct participants receive it. GOD and fairness are equivalent when broadcast channels are accessible [27], which is our setting. However, these two properties are usually analysed under discrete time models which only concern eventual delivery, and thus do not allow quantitative analysis.

Consistent length in Blockchain. Blockchain protocols allow participants to jointly maintain a blockchain. The consistent length property [28, 37–39] of blockchain protocols specify that if a correct participant's blockchain is of length ℓ at time t, then any correct participant's blockchain at time $t+\psi$ is of length at least ℓ. Blockchains trivially satisfy the property in synchronous networks, as a correct participant will send its chain to other correct participants within the synchronous latency Δ.

Adapting the consistent length property from blockchain protocols to DRBs suffers from two limitations. First, it only considers the advantage between correct participants rather than the advantage of an adversary. In particular, the adversary may grow its blockchain faster than correct participants, while withholding its blockchain. This makes it a liveness property rather than a security property. Second, it only concerns the time advantage, i.e., how much time the adversary learns blocks earlier than correct participants, but does not concern how many blocks the adversary can learn earlier than correct participants.

1.2 Our contributions

In this paper, we initiate the study of *delivery-fairness*, a new security property capturing the advantage that some participants learn random outputs earlier than other participants in DRBs. We formalise delivery-fairness, prove its lower bound, and analyse the delivery-fairness guarantee of state-of-the-art DRBs, including Drand [6], HydRand [41], GRandPiper [22] and SPURT [29]. Through the analysis, we identify attacks on delivery-fairness and obtain several insights for improving delivery-fairness. The insights allow us to suggest lock-step variants for HydRand and SPURT with better delivery-fairness (where SPURT achieves the optimal value), without affecting system models or security properties. Table 1 summarises our results.

Table 1: Summary of evaluation results under synchronous networks.

	Protocol	No DKG	Fault tolerance	Comm. compl.		Latency		Delivery-fairness¶	
				Best	Worst	Best	Worst	ω	ψ
Existing work	Drand [6]	✗	$n=2f+1$	$O(n^2)$	$O(n^2)$	δ	Δ	$\frac{\Delta}{\delta}-1$	$\Delta-\delta$
	Lock-step Drand [7]	✗	$n=2f+1$	$O(n^2)$	$O(n^2)$	Δ	Δ	1	$\Delta-\delta$
	HydRand [41]	✓	$n=3f+1$	$O(n^2)$	$O(n^3)^\dagger$	3δ	3Δ	$\frac{\Delta}{3\delta}+f$	$(3f+1)\Delta-\delta$
	GRandPiper [22]	✓	$n=2f+1$	$O(n^2)$	$O(n^2)$	11Δ	11Δ	$f+1$	$(11f+1)\Delta-\delta$
	SPURT [29]	✓	$n=3f+1$	$O(n^2)$	$O(n^2)$	7δ	$(f+7)\Delta^*$	$\frac{\Delta}{7\delta}$	$\Delta-\delta$
This paper	Lock-step HydRand	✓	$n=3f+1$	$O(n^2)$	$O(n^3)^\dagger$	3Δ	3Δ	$f+1$	$(3f+1)\Delta-\delta$
	Lock-step SPURT	✓	$n=3f+1$	$O(n^2)$	$O(n^2)$	7Δ	$(f+7)\Delta^*$	1	$\Delta-\delta$

¶ In (ω,ψ)-delivery-fairness (Definition 5), the delivery-fairness is better when ω and ψ are smaller. When $\omega=1$ and $\psi=\Delta-\delta$, the delivery-fairness is optimal, where δ and Δ are the actual network latency and the latency upper bound, respectively. In practice, $\delta<<\Delta$.

† In the worst case, the adversary does not reveal its committed secrets for f consecutive epochs. In the next epoch, the correct leader needs to broadcast f $O(n)$-size recovery certificates to all participants, leading to $O(n^3)$ communication complexity.

$*$ In the worst case, the adversary controls f consecutive leaders and aborts these f consecutive epochs before a correct epoch with 7Δ, leading to $(f+7)\Delta$ worst-case latency.

Delivery-fairness and its lower bound. We base our study on existing DRB models [22, 29, 31]. As specified in §2, we consider a fixed set of n participants and an adversary who can corrupt up to f of them, where f is protocol-specific. The network is synchronous, where messages are delivered in at least the actual network latency δ and at most a known upper bound Δ. Participants jointly execute the DRB protocol to agree on a unique ledger containing a sequence of random outputs securing three properties, namely consistency, liveness and unpredictability.

Atop the DRB model, we provide the first formal definition of delivery-fairness in §3. The delivery-fairness concerns two aspects of advantage, namely the *length-advantage*, i.e., how many random outputs an adversary can learn earlier than correct participants, and the *time-advantage*, i.e., how much time an adversary can learn a given random output earlier than correct participants.

Definition 1 (Delivery-fairness, informal; formalised in Definition 5). *A DRB protocol satisfies (ω,ψ)-delivery-fairness if the following holds for any two participants (p_i,p_j) and any time t, except for negligible probability:*

- *ω-**length-advantage**: at time t, p_i's ledger pruning the last ω random outputs precedes or is equal to p_j's ledger; and*
- *ψ-**time-advantage**: p_i's ledger at time t precedes or is equal to p_j's ledger at time $t+\psi$.*

When ω and ψ are smaller, the length-advantage and time-advantage of any participant over the other participants are smaller, thus the DRB provides stronger delivery-fairness guarantee. We stress that delivery-fairness is achievable in synchronous networks, where messages are delivered in at least the actual network latency δ and at most a known upper bound Δ. Otherwise, the adversary can arbitrarily delay messages in asynchronous networks or the asynchrony

period in partially synchronous networks, increasing ω and ψ to values that are impractical.

We then prove the lower bound of delivery-fairness in synchronous networks, where ω and ψ are at least 1 and $\Delta-\delta$, respectively. The intuition behind the proof is that, if the time difference of learning a random output between any two participants is smaller than $\Delta-\delta$, then the group of Byzantine participants can produce valid random outputs without communicating with correct participants, contradicting to unpredictability or consistency.

Theorem 1 (Delivery-fairness lower bound, informal; formalised in Theorem 2). *There does not exist a secure DRB protocol that achieves (ω, ψ)-delivery fairness with $\omega < 1$ or $\psi < \Delta-\delta$ under synchronous networks.*

Analysis of delivery-fairness of existing DRBs. With the formalisation, we analyse the delivery-fairness of state-of-the-art DRB protocols, namely Drand [6], HydRand [41], GRandPiper [22] and SPURT [29], in §4. Table 1 summarises the results. Through the analysis, we identify attacks on delivery-fairness and obtain several insights for improving delivery-fairness. Specifically, we identify a new attack called *latency manipulation attack* that can weaken the delivery-fairness in the original versions of Drand, HydRand and SPURT. This attack is rooted in their non-lock-step design that participants can make progress once receiving sufficient messages, without the need of waiting for a fixed time period. Following this observation, we suggest lock-step variants for HydRand and SPURT that resist against the latency manipulation attack and thus achieve the better delivery-fairness (where SPURT achieves the optimal value), without affecting system models or security properties. In addition, a previously known unpredictability-focused attack, which we call *private beacon attack*, can also weaken the delivery-fairness of HydRand and GRandPiper. The private beacon attack is rooted in their design that the epoch leader solely samples the entropy for the random output. To resist against the attack, the entropy should instead be sampled by a group of at least $f+1$ participants, where f is the number of Byzantine participants.

Implications of our results. Our analysis suggests that fair delivery of random outputs is a practical concern for DRB-dependent protocols and applications, in terms of both deployment environment and latency. First, when the security or fairness of a protocol depends on the delivery-fairness of its underlying DRB, then it has to be deployed under a synchronous network where practical delivery-fairness is achievable. Second, DRB-dependent protocols require a random output to be "future enough" such that it remains unknown to everyone at a given time. Our defined delivery-fairness quantifies the minimum waiting time of a future random output, affecting the latency of the DRB-dependent protocols.

2 Model

2.1 System model

Participants. We consider a fixed number of n participants. Each participant $p_k \in [p_n]$ generates a pair of secret key and public key (sk_k, pk_k), and is uniquely identified by its public key in the system. We assume each participant has the knowledge of other participants' public keys.

Adversary. We consider a static adversary $\mathcal{A}$. In the beginning of the protocol, $\mathcal{A}$ can corrupt at most f participants, where f is a corruption parameter subjected to the protocol design. After that, $\mathcal{A}$ cannot change the set of corrupted participants or corrupt more participants. $\mathcal{A}$ fully controls corrupted participants, including observing the participant's internal state and controlling its messages and outputs, without any latency. $\mathcal{A}$ can read all messages between participants, but cannot modify or drop messages sent by correct participants. We also refer to a corrupted participant as *Byzantine* participant. We assume $\mathcal{A}$ is probabilistically polynomial-time (PPT), and thus cannot break standard cryptographic primitives.

Network model. We assume synchronous networks: $\mathcal{A}$ can decide to deliver any message in at least the actual network delay δ and at most a known upper bound Δ. In practice, $\delta << \Delta$.

We will conduct analysis assuming synchronous networks for all DRBs, although some of them can work in relaxed network models. The reason is that the delivery-fairness is a concrete measure and is meaningful only in synchronous networks. Otherwise, in asynchronous networks or the asynchrony period in partially synchronous networks, the adversary can arbitrarily delay messages to increase its advantage, leading to impractical delivery-fairness guarantee. Consequently, applications that require time-sensitive random outputs will be insecure or unfair. Thus, when the application scenarios demand a delivery-fair DRB, the application and DRB have to be deployed in synchronous networks.

2.2 Components of DRBs

The set of n participants continuously execute the DRB protocol Π to produce a sequence of random outputs. Specifically, participants jointly produce and agree on a ledger formed as a sequence of blocks. Each agreed block has to meet a verification predicate. Each block deterministically derives a random output, which can be extracted via a random output extraction function. The verification predicate and random output extraction function are accessible to anyone inside and outside the system, and their instantiation depends on the concrete protocol design.

Ledger. A ledger T is formed as a sequence of blocks. Let $T[e]$ be the e-th block in the ledger T. Let $|T|$ be the length of ledger T. Let $T[p:q]$ be the ledger from p-th block to $(q-1)$-th block of ledger T. Parameter p and q can be set empty, indicating the beginning and the end of the ledger, respectively. Let $T^{\lceil \ell} = T[:-\ell]$

be the ledger from pruning the last ℓ blocks of ledger T. We denote a ledger T is a prefix of or equals to another ledger T' as $T \preceq T'$.

Epoch. DRBs are executed in epochs. In each epoch, participants are expected to produce and agree on a new block. The time period of an epoch can be fixed by the protocol design, or be variant depending on Byzantine behaviours and/or actual network delay. In leader-based DRBs, in each epoch, a leader is elected to drive the protocol execution. In some leader-based DRBs (e.g., SPURT [29]), a Byzantine epoch leader can abort the protocol, so that no block is produced in that epoch.

Verification predicate. To be agreed, a block has to meet the verification predicate $F_V(\cdot)$. In $F_V(\kappa, T, B) \rightarrow \{0,1\}$, given security parameter κ, ledger T and block B as input, outputs 1 if B is a valid successor block of T. A ledger T is valid in κ if for all $\ell \in [|T|-1]$, $F_V(\kappa, T[:\ell], T[\ell]) = 1$. Let $\mathcal{T}_i^t$ be participant p_i's longest valid ledger at time t.

Random output extraction function. Each block contains a random output, which can be extracted by the random output extraction function $F_R(\cdot)$. In $F_R(\kappa, T) \rightarrow R_e$, given security parameter κ and a ledger T of length $|T| = (e-1)$ as input, $F_R(\kappa, T)$ can derive a random output R_e. That is, every block B_e is associated with a random output R_e.

2.3 Security properties of DRBs

A DRB protocol Π should satisfy the following properties, namely consistency, liveness, and unpredictability.

Consistency. Consistency ensures that correct participants agree on a unique ledger, and thus a unique sequence of random outputs. The consistency definition follows the common prefix property in blockchain protocols [32], where the ledgers of any two correct participants are same except for the last ℓ blocks.

Definition 2 (ℓ-consistency, from [32]). *For any κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two correct participants p_i and p_j ($i = j$ is possible) at time t,*

$$(\mathcal{T}_i^t)^{\lceil \ell} \preceq \mathcal{T}_j^t \vee (\mathcal{T}_j^t)^{\lceil \ell} \preceq \mathcal{T}_i^t$$

Liveness. Liveness ensures that correct participants produce new random outputs at an admissible rate. The liveness definition follows the chain growth property in blockchain protocols [34], where for any period of t time a correct participant's ledger grows at least τ blocks.

Definition 3 ((t,τ)-liveness, from [34]). *For any κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any correct participant p_i and time $t' \geq t$,*

$$|\mathcal{T}_i^{t'}| - |\mathcal{T}_i^{t'-t}| \geq t \cdot \tau$$

Unpredictability. Each random output should be *unpredictable*: given an agreed ledger, the adversary cannot predict the next random output before it is produced. If the adversary can predict future random outputs, then it may take advantage in randomness-based applications. The unpredictability definition follows the paradigm that without protocol transcripts from correct participants, no adversary can distinguish between a future random output of the DRB and a randomly sampled string of the same length [29, 31].

Definition 4 (Unpredictability, from [29]). *A DRB protocol Π is unpredictable if for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds. Assuming all participants have agreed on a ledger of e consecutive random outputs $R_1,...,R_e$. For any future random output $R_{e'}$ where $e' > e$ and any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, if $\mathcal{A}$ does not have the knowledge of protocol transcripts associated with $R_{e'}$ from correct participants, then*

$$|\Pr[\mathcal{A}(R_{e'}) = 1] - \Pr[\mathcal{A}(r) = 1]| \leq \mathsf{negl}(\kappa)$$

, where r is a randomly sampled κ-bit string, and $\mathcal{A}(x) \to \{0,1\}$ outputs 1 if $\mathcal{A}$ guesses x to be the random output in epoch e' and otherwise 0.

2.4 Performance metrics

DRBs concern two performance metrics, namely communication complexity and latency.

Communication complexity. Communication complexity is the total amount of communication required to complete a protocol [43]. In DRBs, the communication complexity is quantified as the amount of bits transferred among participants for generating a random output. A protocol may have different communication complexity in the best-case and worst-case executions.

Latency. Latency is the time required to complete a protocol. In the context of DRBs, the latency is quantified as the time participants take to generate a random output. Similarly, a protocol may have different latencies in the best-case and worst-case executions.

3 Delivery-fairness property

In this section, we formally define the delivery-fairness property. The delivery-fairness concerns two aspects of the advantage: *length-advantage* and *time-advantage*. Length-advantage concerns how many random outputs the adversary can learn earlier than correct participants. Time-advantage concerns how much time the adversary can learn a random output earlier than correct participants. We also prove the lower bound of the delivery-fairness, representing the optimal guarantee.

3.1 Defining delivery-fairness

We define delivery-fairness through two strawman definitions that are intuitive but incomplete. We begin with the *fairness* notion in multiparty computation (MPC) protocols that, if the adversary receives the output, then correct participants eventually receive the output [40]. We then generalise the fairness notion to the continuous time model, making it consistent with the DRB settings.

Attempt #1: Time advantage. We first consider relaxing the round-based fairness definition to the continuous time model by introducing a time parameter ψ. Namely, if a participant learns a random output at time t, then all other participants learn this random output no later than time $t+\psi$. However, this definition fails to capture that the adversary may learn more than one random outputs in advance than correct participants.

Attempt #2: Length and time advantage. We then consider capturing both length and time advantage. Let ω be the length parameter and ψ be the time advantage parameter. A DRB protocol satisfies (ω,ψ)-delivery-fairness if for any two participants p_i,p_j: 1) p_i's ledger is longer than p_j's ledger by no more than ω random outputs, and 2) p_j's ledger at time $t+\psi$ is no shorter than p_i's ledger at time t.

However, the definition does not specify whether the last ω random outputs of p_j at time $t+\psi$ should be identical to the last ω random outputs of p_i at time t or not. If not, then this contradicts to the consistency property.

Final definition: Length and time advantage with consistency. We then add the consistency guarantee to the definition in attempt #2, leading to our final definition. Specifically, delivery-fairness concerns the adversary's length advantage and time advantage, parameterised by ω and ψ, respectively. The ω-length-advantage states that the longest valid ledger pruning the last ω blocks is a prefix of the valid ledger in any participant's view at any time. The ψ-time-advantage states that the shortest valid ledger at time t should "catch up with" all participants' ledgers at time t after the time period of ψ. When ω and ψ are smaller, the DRB provides stronger delivery-fairness guarantee.

Definition 5 ((ω,ψ)-Delivery-Fairness). *A DRB protocol Π satisfies (ω,ψ)-delivery-fairness if for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds for any two participants p_i,p_j and any time t except for probability $\mathsf{negl}(\kappa)$:*
- *ω-**length-advantage**: $(\mathcal{T}_i^t)^{\lceil\omega} \preceq \mathcal{T}_j^t$*
- *ψ-**time-advantage**: $\mathcal{T}_i^t \preceq \mathcal{T}_j^{t+\psi}$*

3.2 Lower bound of delivery-fairness

We prove that $(1,\Delta-\delta)$-delivery-fairness is the optimal delivery-fairness guarantee. Specifically, we prove the following theorem.

Theorem 2 (Delivery-fairness lower bound of DRB). *There does not exist a DRB protocol that simultaneously satisfies the following in synchronous networks:*

– *consistency, liveness and unpredictability as in §2; and*
– *(ω,ψ)-delivery fairness with $\omega<1$ or $\psi<\Delta-\delta$*

Proof. Assuming such a DRB protocol exists. Assuming at time t, all participants have agreed on a ledger of e consecutive random outputs $R_1,...,R_e$, and start producing R_{e+1}. $\mathcal{A}$ sets the latency among correct participants to be Δ, the latency of messages from any corrupted participant to any correct participant to be Δ, and the latency of messages from any correct participant to any corrupted participant to be δ. By unpredictability, without messages from correct participants, corrupted participants cannot learn the value of R_{e+1}. Thus, the fastest possible way for corrupted participants to learn a random output is to get messages from correct participants, which is at least $t+\delta$. Given the latency set by $\mathcal{A}$, a correct participant receives messages only at $t+\Delta$.

By the assumption that the time-advantage between correct and corrupted participants is smaller than $\Delta-\delta$, correct participants learn the random output R_{e+1} before $t+\Delta$. Thus, a correct participant has to learn the random output R'_{e+1} that satisfies the verification predicate. If $R'_{e+1}=R_{e+1}$, then this means that a participant can solely produce random outputs without interacting with the other participants. Thus, f corrupted participants can also produce random outputs without interacting with the other correct participants, contradicting to the unpredictability property. If $R'_{e+1}\neq R_{e+1}$, then this means that f corrupted participants can produce valid random outputs conflicted with those from the other participants, contradicting to the consistency property.

4 Delivery-fairness analysis of existing DRBs

In this section, we provide the delivery-fairness analysis of state-of-the-art DRB protocols, namely Drand [6], HydRand [41], GRandPiper [22] and SPURT [29]. Table 1 summarises the results. Appendix B provides the full specifications and analysis of the DRBs.

4.1 Drand

Summary of design. Drand a DRB protocol based on the BLS threshold signature [24]. It allows a threshold number of participants in a group to jointly sign a message, where the signature is publicly verifiable. Appendix A.1 summarises the syntax of BLS threshold signature.

In Drand design, participants perform a distributed key generation (DKG) to generate secret keys, and agree on an initial signature σ^0. Then, for each epoch e, participants jointly generate a BLS threshold signature σ^e over e and the last epoch's BLS signature σ^{e-1} (or σ^0 at the first epoch). An epoch's random output R_e is the calculated as $H(\sigma^e)$, where $H(\cdot)$ is a hash function. Drand requires a DKG due to the usage of threshold signature, and achieves the fault tolerance capacity of $n=2f+1$. Appendix B.1 provides the full specification of Drand.

In the beginning of the DRB protocol, the adversary $\mathcal{A}$ does the follows.

1. $\mathcal{A}$ chooses $n-2f$ correct participants.
2. $\mathcal{A}$ sets the latency of messages among f corrupted participants and $n-2f$ correct participants to be δ.
3. $\mathcal{A}$ sets the latency of messages from, to and among the other f correct participants to be Δ.

Fig. 1: Latency manipulation attack on leaderless DRBs.

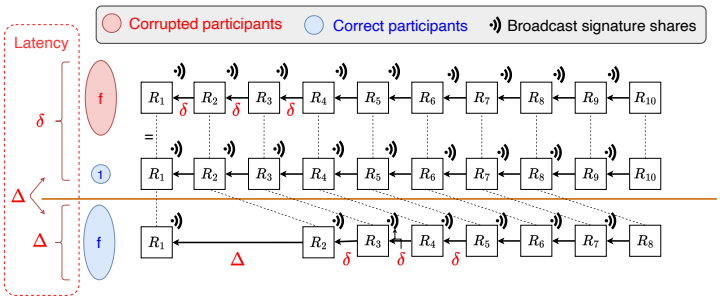

Fig. 2: The latency manipulation attack on the non-lock-step Drand Π_{Drand} with $\Delta = 3\delta$. The adversary $\mathcal{A}$ chooses $n-2f=1$ correct participant, sets the latency among f corrupted participants and the chosen correct participant (above the brown horizontal line) to be δ, and sets the latency from, to, and among other f correct participants (below the brown horizontal line) to be Δ. During the first Δ, participants above and below the brown horizontal line learn $\frac{\Delta}{\delta}+1=4$ and 2 random outputs, respectively, leading to length-advantage degree $\omega=4-2=2$. Note that in real-world networks, $\Delta >> \delta$ and ω could be much larger than in this example. For each of these random outputs, participants above the brown line learn it earlier than those below the brown line by $\Delta-\delta$, leading to time-advantage degree $\psi=\Delta-\delta$.

Drand has two variants, namely the non-lock-step Π_{Drand} specified in the documentation [8] and the lock-step $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ in the actual implementation [7]. Compared to Π_{Drand}, $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ requires participants to wait for a time period during the phase of broadcasting signatures for each epoch e. In synchronous networks, the time period is at least Δ. In Drand's implementation $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ [7], the default time period (named `DefaultBeaconPeriod`) is 60 seconds.

Latency manipulation attack. We identify a new attack *latency manipulation attack* that can increase the adversary's advantage of delivery-fairness in the non-lock-step Π_{Drand}. The latency manipulation attack only requires the adversary to manipulate the latency among participants (subjected to the network model), and does not require equivocating or withholding messages. Thus, the attack is unaccountable and does not affect other security properties or performance metrics.

Figure 1 presents the latency manipulation attack on leaderless DRBs, e.g., Drand. Figure 2 depicts an example latency manipulation attack on the non-lock-step Drand Π_{Drand}. The adversary $\mathcal{A}$ follows the protocol with $n-2f$ (which is 1 in Drand) correct participants while delaying all messages from and to the other

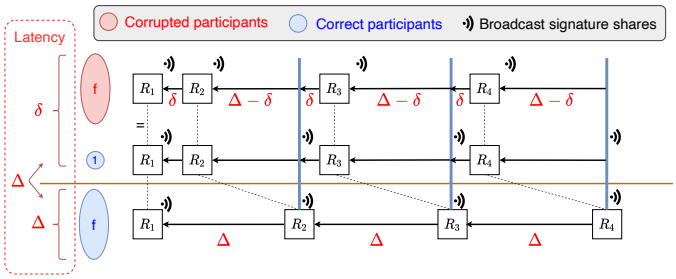

Fig. 3: Example of the latency manipulation attack on the lock-step Drand $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ with $\Delta = 3\delta$. While corrupted participants will enter the next epoch immediately after learning a random output, correct participants will stay in every epoch for Δ, even learning the random output of this epoch in advance (i.e., δ since the beginning of the epoch). Consequently, $\mathcal{A}$ can only gain $(1,\Delta-\delta)$-delivery-fairness.

f correct participants. During the first Δ under latency manipulation attack, $\mathcal{A}$ and $n-2f$ correct participants learn a random output for every δ, while the other f correct participants learn a random output only at the end of this Δ.

Delivery-fairness of non-lock-step Drand. At the end of this Δ, $\mathcal{A}$ and $n-2f$ correct participants have learned $\frac{\Delta}{\delta}+1$ random outputs, while the other f correct participants only learn two random outputs, leading to length-advantage degree $\omega = \frac{\Delta}{\delta} - 1$. In real-world networks, $\Delta >> \delta$, which leads to large value of ω. For each of these random outputs (except for the first one), $\mathcal{A}$ and $n-2f$ correct participants learn it earlier than the other f correct participants by $\Delta-\delta$, leading to time-advantage degree $\psi = \Delta-\delta$. Therefore, the non-lock-step Drand Π_{Drand} achieves (ω,ψ)-delivery-fairness where $\omega = \frac{\Delta}{\delta} - 1$ and $\psi = \Delta-\delta$.

Delivery-fairness of lock-step Drand. We analyse the delivery-fairness guarantee of $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$, and show that $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ achieves optimal $(1,\Delta-\delta)$-delivery-fairness. The improvement compared to the non-lock-step Π_{Drand} is due to the lock-step design, where correct participants will wait for Δ before entering the next epoch and broadcasting signature shares, even learning the random output of this epoch in advance, as depicted in Figure 3. Appendix B.1 provides the full analysis.

Gained insights. Through the analysis, we obtain an insight on improving the delivery-fairness. Namely, the lock-step execution is necessary to bound the adversary's length-advantage to 1. Otherwise, without the lock-step execution, the latency manipulation attack can always allow the adversary to grow its ledger faster than correct participants within a Δ, and thus increase its advantage in length and time.

4.2 HydRand and GRandPiper

We analyse the delivery-fairness of HydRand [41] and GRandPiper [22], two DRB protocols based on the rotating leader paradigm and PVSS. We observe that a previously known unpredictability-focused attack, which we call *private beacon attack*, weakens the delivery-fairness of HydRand and GrandPiper. The

attack is rooted in their design is that the entropy of a random output is solely provided by the epoch leader. To resist against the attack, the entropy should instead be provided by a group of at least $f+1$ participants.

Summary of HydRand. HydRand is a DRB protocol based on leader election, accumulator and publicly verifiable secret sharing (PVSS) [26, 42]. Leader election allows a group of participants to elect a leader for every epoch. Accumulator [20] allows to compress a set of values into a short accumulation value, and prove the inclusion of each value given the accumulation value and a short witness. PVSS [26, 42] allows one to distribute a secret with a group of participants, in which a threshold number of participants can collaboratively reconstruct the secret. Their syntaxes are summarised in Appendix A.2-A.4.

In HydRand design, participants employ the round-robin leader election to elect a leader for every epoch. In each epoch, the leader solely samples a random input, generates the commitment of this random input, uses PVSS to generate secret shares for this random input, and use the accumulator to generate an accumulation value for these secret shares. Then, the leader broadcasts the commitment, a secret share, and the accumulation value to each participant. Meanwhile, the leader can choose whether to reveal its last random input, which, together with the last f random outputs, determines this epoch's random output. If the leader is Byzantine and does not reveal it, then all participants reconstruct the last random input via broadcasting secret shares. HydRand does not require distributed key generation and achieves the fault tolerance capacity of $n=3f+1$. Appendix B.2 provides the full specification of HydRand.

The original HydRand protocol Π_{HydRand} in the paper [41] and implementation [9] is non-lock-step. We also study its lock-step variant $\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ that resists against the latency manipulation attacks and achieves better delivery-fairness.

In the beginning of the DRB protocol, the adversary $\mathcal{A}$ does the follows.

1. $\mathcal{A}$ chooses $n-2f$ correct participants.
2. $\mathcal{A}$ sets the latency of messages among f corrupted participants and $n-2f$ correct participants to be δ.
3. $\mathcal{A}$ sets the latency of messages from, to and among the other f correct participants to be Δ.
4. Upon a new random output R_e, $\mathcal{A}$ calculates the next leader l_{e+1} based on Π_{LE}. If l_{e+1} is the participant with δ latency, then keep running the attack, otherwise stop the attack.

Fig. 4: Latency manipulation attack on leader-based DRBs. Extra specification compared to the latency manipulation attack on leaderless DRBs is labelled in blue.

Latency manipulation attack on leader-based DRBs. Recall that the latency manipulation attack on leaderless DRBs (e.g., Drand) allows the adversary $\mathcal{A}$ to learn random outputs faster than f correct participants with Δ latency,

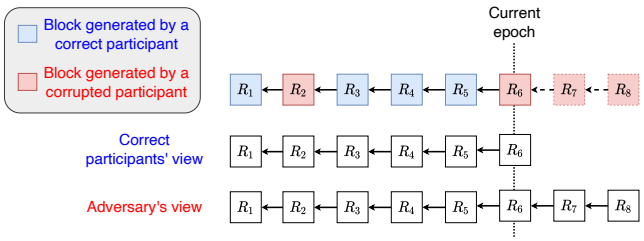

Fig. 5: Example of a private beacon attack on the lock-step HydRand $\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$. Assuming at the current epoch $e=6$, the leader l_6 is a corrupted participant. Leader l_6 reveals its committed secret and calculates the current random output R_6, which determines the next epoch's leader l_7, and so on. When l_6, l_7 and l_8 are all corrupted participants, the adversary $\mathcal{A}$ can learn R_7 and R_8 when epoch $e=6$, weakening the delivery-fairness.

within a time period of Δ. However, the latency manipulation attack in leader-based DRBs faces a different scenario: during this Δ, if one of these f correct participants becomes the leader in an epoch, then $\mathcal{A}$ cannot learn any new random output until this leader reaches this epoch. Consequently, $\mathcal{A}$ has to stop the attack, leading to less advantage compared to the attack in leaderless protocols like Drand. Figure 4 presents the latency manipulation attack on leader-based DRBs.

Private beacon attack. Bhat et al. [22] observes an attack on the unpredictability of HydRand and GRandPiper. This attack, which we call *private beacon attack*, can also weaken the delivery-fairness of HydRand (including both Π_{HydRand} and $\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$) and GRandPiper. In this attack, the adversary grows its own ledger to learn random outputs earlier than correct participants. As HydRand allows the epoch leader to solely sample the entropy, the epoch leader can learn the random output instantly without communicating with others. When c consecutive leaders are corrupted, the adversary can learn c future random outputs. With round-robin leader election used in HydRand, GRandPiper and SPURT, c is at most f, as analysed in Appendix A.2. Same as the latency manipulation attack, the private beacon attack does not require equivocating or withholding messages, and thus remains unaccountable. The private beacon attack is presented in Figure 6 and depicted in Figure 5.

While following the DRB protocol, the adversary $\mathcal{A}$ does the follows.

1. Upon a new random output R_e, $\mathcal{A}$ calculates the next leader l_{e+1} based on Π_{LE}.
2. If the next leader l_{e+1} is a Byzantine participant, $\mathcal{A}$ follows the protocol to sample the random output R_{e+1} locally and repeats step 1.

Fig. 6: Private beacon attack on DRBs.

Delivery-fairness of non-lock-step HydRand. Both the latency manipulation attack and the private beacon attack can be applied to the non-lock-

step Π_{HydRand}. Under both attacks, the non-lock-step Hydrand Π_{HydRand} achieves (ω,ψ)-delivery-fairness where $\omega = \frac{\Delta}{3\delta} + f$ and $\psi = (3f+1)\Delta - \delta$. Appendix B.2 provides the full analysis.

Delivery-fairness of lock-step HydRand. The lock-step execution rules out the latency manipulation attack. Under the private beacon attack, the lock-step HydRand $\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ achieves (ω,ψ)-delivery-fairness where $\omega = f+1$ and $\psi = (3f+1)\Delta - \delta$. Appendix B.2 provides the full analysis.

Summary of GRandPiper. GRandPiper is a DRB based on leader election, Byzantine broadcast and publicly verifiable secret sharing (PVSS). Byzantine broadcast Π_{BB} allows a designated broadcaster broadcasts a value to a group of participants, such that all correct participants will commit the same value. If the broadcaster is correct, then all correct participants will commit the broadcasted value. Appendix A.5 provides its definition. GRandPiper employs a Byzantine broadcast protocol with $O(n^2)$ communication complexity and latency $t_{\mathsf{BB}} = 11\Delta$.

GRandPiper [22] follows the HydRand's approach with three major modifications. First, GRandPiper enforces participants to recover the secret random input committed by the leader, without allowing the leader to reveal it by itself. Second, GRandPiper replaces the Acknowledge and Vote-confirm phase in HydRand with an explicit Byzantine broadcast protocol Π_{BB}. Note that the Byzantine broadcast and the round-robin leader election constitute a SMR protocol, as described in the RandPiper paper. Third, GRandPiper formalises the Hydrand's idea of separating the process of committing and revealing random inputs as a queue-based mechanism, where each participant buffers previously committed secret values and pops one value to reconstruct for each epoch. GRandPiper does not require distributed key generation and achieves the fault tolerance capacity of $n = 2f+1$. Appendix B.3 provides the full specification of GRandPiper.

Delivery-fairness of GRandPiper. GRandPiper is lock-step and thus rules out the latency manipulation attack. Under the private beacon attack, GRandPiper achieves (ω,ψ)-delivery-fairness where $\omega = f+1$ and $\psi = (11f+1)\Delta - \delta$. Appendix B.3 provides the full analysis.

Gained insights. In HydRand and GRandPiper, the entropy of a random output is provided by a sole leader. In this case, the adversary can always launch the private beacon attack as long as the leader is Byzantine. To mitigate the private beacon attack, the protocol should prevent the adversary from controlling the entropy for a random output. To this end, the entropy should instead be provided by a group of at least $f+1$ participants rather than a single participant.

4.3 SPURT

Summary of design. SPURT is a DRB based on leader election, Byzantine broadcast, and a specialised PVSS protocol $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}$. The used Byzantine broadcast protocol a variant of HotStuff [44] with best-case latency of 4δ and worst-case latency $t_{\mathsf{BB}} = 4\Delta$. Appendix A.4 summarises the differences between $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}$ and the traditional Π_{PVSS}.

In the SPURT design, each participant samples a random input, uses PVSS to generate secret shares, encrypted secret shares and inclusion proofs of this random input, and sends all encrypted shares and inclusion proofs to the leader elected via a round-robin leader election. Then, the leader homomorphically aggregates all received commitments and inclusion proofs column-wise, and triggers a Byzantine broadcast over the aggregated encrypted shares and inclusion proofs, such that all participants agree on the entropy for the random output. After Byzantine broadcast, each participant decrypts one of the encrypted shares and broadcasts the decrypted share, such that all participants can reconstruct the secret from received decrypted shares. SPURT [29] does not require distributed key generation and achieves the fault tolerance capacity of $n = 3f + 1$. Appendix B.4 provides the full specification of SPURT.

The original SPURT protocol Π_{SPURT} in the paper [29] and implementation [16] is non-lock-step. We also study its lock-step variant $\Pi^{\mathsf{LS}}_{\mathsf{SPURT}}$ that resists against the latency manipulation attacks and achieves optimal delivery-fairness.

Delivery-fairness of non-lock-step SPURT. SPURT resists against the private beacon attack, as the entropy of a random output is jointly provided by $f+1$ participants. However, similar to HydRand, the non-lock-step Π_{SPURT} does not resist against the latency manipulation attack in Figure 4. Specifically, $\mathcal{A}$ sets the latency among its corrupted participants and $n-2f = f+1$ correct participants as δ, while the latency from, to and among the rest correct participant as Δ. Recall that SPURT achieves the 7δ best-case latency and 7Δ worst-case latency. Similar to non-lock-step HydRand Π_{HydRand}, after a Δ of the latency manipulation attack, $\mathcal{A}$ and $f+1$ correct participants learn $\frac{\Delta}{7\delta}+1$ random outputs, while the rest correct participant only knows a single random output, leading to $\frac{\Delta}{7\delta}$ length-advantage and $\Delta-\delta$ time-advantage. Thus, the non-lock-step Π_{SPURT} achieves (ω,ψ)-delivery-fairness with $\omega = \frac{\Delta}{7\delta}$ and $\psi = \Delta - \delta$.

Delivery-fairness of lock-step SPURT. We then analyse the delivery-fairness guarantee of $\Pi^{\mathsf{LS}}_{\mathsf{SPURT}}$, and show that $\Pi^{\mathsf{LS}}_{\mathsf{SPURT}}$ achieves the optimal $(1,\Delta-\delta)$-delivery-fairness. Appendix B.4 provides the full analysis.

5 Conclusion

In this paper, we have introduced and formalised a new property, named delivery-fairness, for DRB protocols. We proved the lower bound for delivery-fairness, and analysed the delivery-fairness guarantee of state-of-the-art DRB protocols. The analysis reveals insights on ensuring and improving the delivery-fairness guarantee. In particular, when delivery-fairness is necessary for DRB-dependent applications, then the DRB has to be deployed under synchronous networks, otherwise the adversary can arbitrarily delay messages to weaken delivery-fairness during the asynchronous period. In addition, in order to achieve delivery-fairness, the DRB has to follow the lock-step execution to rule out latency manipulation attacks, and allows at least $f+1$ nodes to sample the entropy to rule out the private beacon attacks.

References

1. 16 Ways to Create Dynamic Non-Fungible Tokens (NFT) Using Chainlink Oracles, https://blog.chain.link/create-dynamic-nfts-using-chainlink-oracles/
2. Celo: Mobile-First DeFi Platform for Fast, Secure, and Stable Digital Payments https://celo.org/
3. Chainlink VRF, https://docs.chain.link/docs/chainlink-vrf/
4. CryptOrchids: NFT plants that must be watered weekly, https://cryptorchids.io/
5. Distributed Randomness Beacon | Cloudflare, https://www.cloudflare.com/leagueofentropy/
6. Drand - Distributed Randomness Beacon. https://drand.love/
7. Drand: A Distributed Randomness Beacon Daemon - Go implementation https://github.com/drand/drand
8. Drand Specification https://drand.love/docs/specification
9. HydRand: Python implementation of the HydRand protocol https://github.com/PhilippSchindler/HydRand
10. Join Hermez Trusted Setup Phase 2 Ceremony! https://blog.hermez.io/hermez-trusted-setup-phase-2/
11. PancakeSwap Lottery, https://pancakeswap.finance/lottery
12. Phase 2 setup random beacon of Celo https://github.com/celo-org/celo-bls-snark-rs/issues/227
13. Polygon Hermez https://hermez.io/
14. PolyRoll: Decentralized Games, https://polyroll.org/
15. Reinforcing the Security of the Sapling MPC https://electriccoin.co/blog/reinforcing-the-security-of-the-sapling-mpc/
16. SPURT implementation, forked from HydRand https://github.com/sourav1547/HydRand
17. The Economic Impact of Random Rewards in Blockchain Video Games, https://blog.chain.link/the-economic-impact-of-random-rewards-in-blockchain-video-games/
18. Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Byzantine agreement, broadcast and state machine replication with near-optimal good-case latency (2020)
19. Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Good-case latency of byzantine broadcast: a complete categorization. arXiv preprint arXiv:2102.07240 (2021)
20. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: International conference on the theory and applications of cryptographic techniques. pp. 480–494. Springer (1997)
21. Benet, J., Greco, N.: Filecoin: A decentralized storage network. Protocol Labs pp. 1–36 (2018)
22. Bhat, A., Shrestha, N., Luo, Z., Kate, A., Nayak, K.: Randpiper–reconfiguration-friendly random beacons with quadratic communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 3502–3524 (2021)
23. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: International Workshop on Public Key Cryptography. pp. 31–46. Springer (2003)
24. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)

25. Bowe, S., Gabizon, A., Miers, I.: Scalable multi-party computation for zk-snark parameters in the random beacon model. Cryptology ePrint Archive (2017)
26. Cascudo, I., David, B.: SCRAPE: Scalable randomness attested by public entities. In: International Conference on Applied Cryptography and Network Security. pp. 537–556. Springer (2017)
27. Cohen, R., Lindell, Y.: Fairness versus guaranteed output delivery in secure multiparty computation. Journal of Cryptology **30**(4), 1157–1186 (2017)
28. Daian, P., Pass, R., Shi, E.: Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In: International Conference on Financial Cryptography and Data Security. pp. 23–41. Springer (2019)
29. Das, S., Krishnan, V., Isaac, I.M., Ren, L.: Spurt: Scalable distributed randomness beacon with transparent setup. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 1380–1395. IEEE (2022)
30. David, B., Gaži, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 66–98. Springer (2018)
31. Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: European Symposium on Security and Privacy (2021)
32. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 281–310. Springer (2015)
33. Hanke, T., Movahedi, M., Williams, D.: Dfinity technology overview series, consensus system. arXiv preprint arXiv:1805.04548 (2018)
34. Kiayias, A., Panagiotakos, G.: Speed-security tradeoffs in blockchain protocols. IACR Cryptol. ePrint Arch. **2015**, 1019 (2015)
35. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual International Cryptology Conference. pp. 357–388. Springer (2017)
36. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: 34th International Symposium on Distributed Computing (DISC 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
37. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 643–673. Springer (2017)
38. Pass, R., Shi, E.: Hybrid consensus: Efficient consensus in the permissionless model. In: 31st International Symposium on Distributed Computing (DISC 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
39. Pass, R., Shi, E.: Thunderella: Blockchains with optimistic instant confirmation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 3–33. Springer (2018)
40. Pass, R., Shi, E., Tramer, F.: Formal abstractions for attested execution secure processors. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 260–289. Springer (2017)
41. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: HydRand: Efficient Continuous Distributed Randomness. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 32–48

42. Schoenmakers, B.: A simple publicly verifiable secret sharing scheme and its application to electronic voting. In: Annual International Cryptology Conference. pp. 148–164. Springer (1999)
43. Yao, A.C.C.: Some complexity questions related to distributive computing (preliminary report). In: Proceedings of the eleventh annual ACM symposium on Theory of computing. pp. 209–213 (1979)
44. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 347–356 (2019)

A Primitives

A.1 BLS threshold signature

BLS threshold signature [23] is a threshold version of the BLS signature [24]. A (k,n)-BLS signature allows any k out of n participants to jointly sign a message. It requires a distributed key generation (DKG) protocol, where the set of n participants take security parameter κ and threshold $k \leq n$ as input, then each participant p_i receives a unique secret key sk_i and an identical public key pk. The BLS threshold signature Π_{BLS} consists of the following functions.

- $\mathsf{Sign}(sk_i, m) \to \sigma_i$: On input secret key sk_i and message m, outputs signature share σ_i.
- $\mathsf{Aggregate}(\vec{\boldsymbol{\sigma}}) \to \{\sigma, \perp\}$: On input k different signature shares $\vec{\boldsymbol{\sigma}}$, outputs a signature σ if every signature share $\sigma_i \in \vec{\boldsymbol{\sigma}}$ is correctly signed by sk_i otherwise $\perp$.
- $\mathsf{Verify}(pk, m, \sigma) \to \{0,1\}$: On input public key pk, message m and signature σ, outputs 1 if σ is aggregated from k different correct signature shares otherwise 0.

Threshold BLS signature is *unique*: for any two subsets $(\vec{\boldsymbol{\sigma}}, \vec{\boldsymbol{\sigma'}})$ of k different signature shares in $\{\sigma_i\}_{i\in[n]}$ on a message m, $\Pi_{\mathsf{BLS}}.\mathsf{Aggregate}(\vec{\boldsymbol{\sigma}}) = \Pi_{\mathsf{BLS}}.\mathsf{Aggregate}(\vec{\boldsymbol{\sigma'}})$.

A.2 Leader election

Leader election protocol Π_{LE} allows participants to elect a leader for every epoch. Specifically, given the set of participants and the agreed ledger (recording historical random outputs and leaders) in epoch $e-1$ as input, Π_{LE} outputs a leader l_e for epoch e. Let $X(\Pi_{\mathsf{LE}}, n, f, c)$ be the event that c consecutive leaders are corrupted in Π_{LE}.

HydRand and RandPiper employ a round-robin leader election protocol $\Pi_{\mathsf{LE}}^{\mathsf{RR}}$, where a leader is elected from all participants excluding last f leaders and misbehaving leaders randomly with the last random output. Specifically, assuming the current epoch is e. Let $\mathcal{L}_{\text{last}} \leftarrow \{l_{e-f}, \ldots, l_{e-1}\}$ be the last f leaders. Let $\mathcal{L}_w$ be the set of participants that are removed due to misbehaviours. Let $\mathcal{L}_e \leftarrow \{l_0, \ldots, l_{w-1}\} = [p_n] \setminus (\mathcal{L}_{\text{last}} \bigcup \mathcal{L}_w)$ be the set of candidate leaders of epoch e ordered canonically. The leader l_e of epoch e is $l_e \leftarrow l_{(R_{e-1} \mod w)}$.

Existing analysis [22] has proven that $\Pr[X(\Pi_{\mathsf{LE}}^{\mathsf{RR}}, n, f, c)] = \frac{\binom{f}{c}}{(n-f)^c}$. The best, average, and worst values of c are 0, 2, and f, respectively.

Lemma 1 (from [22]). *Assuming a DRB with n participants, in which f participants are corrupted by the adversary $\mathcal{A}$. Under the round-robin leader election $\Pi_{\mathsf{LE}}^{\mathsf{RR}}$, assuming no participant is removed due to misbehavior, the probability $\Pr[X(\Pi_{\mathsf{LE}}^{\mathsf{RR}},n,f,c)]$ that c consecutive leaders are corrupted is*

$$\Pr[X(\Pi_{\mathsf{LE}}^{\mathsf{RR}},n,f,c)] = \frac{\binom{f}{c}}{(n-f)^c}$$

Proof. Let $\mathcal{L}$ be the vector of c leaders in the last c epochs. For c consecutive epochs, the total number of different outcomes of $\mathcal{L}$ is $(n-f)^c$. Recall in $\Pi_{\mathsf{LE}}^{\mathsf{RR}}$ that a participant cannot be the leader in epoch e if it was once a leader in the last f epochs $(e-f,\ldots,e-1)$. As the system only consists of f Byzantine participants, $X(\Pi_{\mathsf{LE}}^{\mathsf{RR}},n,f,c)$ is possible only when $c \leq f$. Among the $(n-f)^c$ outcomes of $\mathcal{L}$, $\binom{f}{c}$ outcomes lead to $X(\Pi_{\mathsf{LE}}^{\mathsf{RR}},n,f,c)$. Therefore, the probability $\Pr[X(\Pi_{\mathsf{LE}}^{\mathsf{RR}},n,f,c)]$ that $X(\Pi_{\mathsf{LE}}^{\mathsf{RR}},n,f,c)$ happens is $\Pr[X(\Pi_{\mathsf{LE}}^{\mathsf{RR}},n,f,c)] = \frac{\binom{f}{c}}{(n-f)^c}$.

A.3 Accumulator

Accumulator [20] allows to compress a set $\mathcal{D}$ of values into a short accumulation value z. For each value $d_i \in \mathcal{D}$, there is a short witness w_i proving that d_i is one of the values accumulated into z. An accumulator Π_{Acc} consists of the following algorithms:

- $\mathsf{Setup}(\kappa,n) \to k$: On input parameter κ and accumulation threshold n, outputs accumulation key k. Note that n is the upper bound on the total number of values that can be securely accumulated.
- $\mathsf{Eval}(k,\mathcal{D}) \to z$: On input key k and a set $\mathcal{D}$ of values, outputs accumulation value z.
- $\mathsf{CreateWit}(k,z,d_i,\mathcal{D}) \to \{w_i,\perp\}$: On input key k, accumulation value z, value d_i, outputs witness w_i if $d_i \in \mathcal{D}$ otherwise $\perp$.
- $\mathsf{Verify}(k,z,d_i,w_i) \to \{0,1\}$: On input key k, accumulation value z for $\mathcal{D}$, value d_i and witness w_i, outputs 1 if $d_i \in \mathcal{D}$ otherwise 0.

A.4 Publicly verifiable secret sharing

A (k,n)-Public Verifiable Secret Sharing (PVSS) [26,42] allows one to distribute a secret with n parties, in which any k parties can collaboratively reconstruct the secret. A PVSS scheme Π_{PVSS} consists of the following algorithms:

- $\mathsf{Gen}(\kappa) \to (sk,pk)$: on input security parameter κ, outputs key pair (sk,pk).
- $\mathsf{Encrypt}(pk_i,s_i) \to c_i$: on input public key pk_i and share s_i, outputs encrypted share c_i.
- $\mathsf{Decrypt}(sk_i,c_i) \to s_i$: on input secret key sk_i and encrypted share c_i, outputs share s_i. It holds that $s_i = \mathsf{Decrypt}(sk_i,\mathsf{Encrypt}(pk_i,s_i))$.
- $\mathsf{Share}(\{pk_i\}_{i\in[n]},k,s) \to (\vec{\boldsymbol{s}},\vec{\boldsymbol{c}},\vec{\boldsymbol{\pi}})$: on public keys $\{pk_i\}_{i\in[n]}$, threshold k, and secret s, outputs shares $\vec{\boldsymbol{s}} = \{s_i\}_{i\in n}$, encrypted shares $\vec{\boldsymbol{c}} = \{c_i\}_{i\in n} = \{\mathsf{Encrypt}(pk_i,s_i)\}_{i\in n}$, and proofs $\vec{\boldsymbol{\pi}} = \{\pi_i\}_{i\in n}$.

– $\mathsf{Verify}(c_i,\pi_i)\rightarrow\{0,1\}$: on input encrypted share c_i and proof π_i, outputs 1 if c_i is encrypted from a certain valid share otherwise 0. Note that the proof π_i does not reveal which share is associated with c_i.
– $\mathsf{Recon}(\vec{\boldsymbol{s}})\rightarrow s$: on input a set $\vec{\boldsymbol{s}}$ of k valid shares, outputs secret s.

SPURT employs a specialised PVSS protocol $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}$. Π_{BB} is a variant of HotStuff [44] with best-case latency of 4δ and worst-case latency $t_{\mathsf{BB}} = 4\Delta$. $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}$ differs from the traditional Π_{PVSS} in three aspects. First, $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}.\mathsf{Recon}(\cdot)$ outputs $e(h_0^s,h_1)$ rather than secret s itself, where $e(\cdot)$ is a bilinear pairing function and (h_0,h_1) are public parameters. Second, an encrypted share in $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}$ consists of two elements (v_i,c_i). Last, proof π_i in $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}.\mathsf{Share}(\cdot)/\mathsf{Verify}(\cdot)$ is omitted in certain scenarios for better performance. To keep notations consistent, we denote (v_i,c_i) in $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}$ as a single element c_i, and explicitly include π_i in $\Pi^{\mathsf{uniform}}_{\mathsf{PVSS}}.\mathsf{Verify}(\cdot)$.

A.5 Byzantine broadcast

In Byzantine broadcast Π_{BB}, a designated broadcaster in a set of n participants broadcasts a value m to other participants such that the following holds [18,19,36]:

– **Agreement:** If two correct participants commit values v and v' respectively, then $v=v'$.
– **Termination:** All correct participants eventually commit a value.
– **Validity:** If the designated broadcaster is correct, then all correct participants commit m.

By executing the leader election and allowing the leader to launch a Byzantine broadcast protocol for each epoch, one can yield a *state machine replication (SMR)* protocol. In SMR, participants commit client requests as a linearisable ledger (aka log) with two guarantees, namely *consistency* that every two correct participants commit the same value at the same ledger position and *liveness* that every client request is eventually committed by all correct participants [18].

B Full specification and analysis

B.1 Drand

Specification. Figure 7 outlines the specification of Π_{Drand} and $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ from the perspective of participant p_i in epoch e.

Delivery-fairness of lock-step Drand. We analyse the delivery-fairness guarantee of $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$, and show that $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ achieves optimal $(1,\Delta-\delta)$-delivery-fairness.

Lemma 2 ($\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ epoch execution). *In $\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$, at the end of every epoch e, i.e., $t=e\cdot\Delta$, for any two participants (p_i,p_j), $\mathcal{T}_i^t=\mathcal{T}_j^t$.*

1. **(Setup)** All participants jointly complete the one-time setup as follows.
   (a) All participants participate in the DKG protocol with security parameter κ and threshold $k = f + 1$ as input, so that each participant p_i obtains the public key pk and a secret key sk_i.
   (b) All participants agree on the initial unique signature σ^0.
2. **(Broadcast signature)** Upon the signature σ^{e-1} in epoch $e-1$ from others or itself, participant p_i does the following.
   (a) p_i executes $\Pi_{\mathsf{BLS}}.\mathsf{Verify}(pk, H((e-1)\|\sigma^{e-2}), \sigma^{e-1})$ to verify whether σ^{e-1} is valid.
   (b) p_i executes $\sigma_i^e \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Sign}(sk_i, H(e\|\sigma^{e-1}))$ to generate signature share σ_i^e.
   (c) p_i broadcasts σ_i^e.
   (d) p_i sets a timer Δ.
3. **(Generate random output)** Upon timer Δ expires and receiving at least $f+1$ different signature shares in epoch $e-1$, participant p_i does the following.
   (a) p_i executes $\sigma^e \leftarrow \Pi_{\mathsf{BLS}}.\mathsf{Aggregate}(\{\sigma_i^e\})$ to obtain the aggregated signature σ^e.
   (b) p_i broadcasts σ^e.
   (c) p_i calculates the random output $R_e \leftarrow H(\sigma^e)$.

Fig. 7: Specification of Drand Π_{Drand}. Extra specification of its lock-step variant $\Pi_{\mathsf{Drand}}^{\mathsf{LS}}$ is labelled in blue.

Proof. The proof is by induction: given the base case where all participants start executing the protocol, assuming an induction hypothesis holds at the end of epoch $e-1$, we prove the induction step that the hypothesis holds at the end of epoch e. The proof implies that the induction hypothesis holds at the end of every epoch.

- **Bese case:** At time $t=0$, for any two participants (p_i,p_j), $\mathcal{T}_i^0 = \mathcal{T}_j^0$.
- **Induction hypothesis:** At time $t=(e-1)\Delta$, for any two participants (p_i,p_j), $\mathcal{T}_i^t = \mathcal{T}_j^t$.
- **Proof goal:** At time $t=e\cdot\Delta$, for any two participants (p_i,p_j), $\mathcal{T}_i^t = \mathcal{T}_j^t$.

The induction step is as follows. After $(e-1)\cdot\Delta$, $n-f$ participants sign $(e\|\sigma^{e-1})$ and broadcast their signatures to each other. No later than $t=(e-1)\cdot\Delta+\Delta<e\cdot\Delta$, $n-f$ participants will receive $n-f$ signatures and can reconstruct σ^e, which leads to $\mathcal{T}_i^t = \mathcal{T}_j^t$ and thus closes the induction proof.

Lemma 3 ($\Pi_{\mathsf{Drand}}^{\mathsf{LS}}$ length-advantage). *$\Pi_{\mathsf{Drand}}^{\mathsf{LS}}$ achieves 1-length-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $(\mathcal{T}_i^t)^{\lceil 1} \preceq \mathcal{T}_j^t$.*

Proof. By Lemma 2, at the end of every epoch e, i.e., $t = e\cdot\Delta$, for any two participants (p_i,p_j), $\mathcal{T}_i^t = \mathcal{T}_j^t$. Therefore, for any $t\in((e-1)\cdot\Delta,e\cdot\Delta]$, participants only differ in σ^e, leading to 1-length-advantage.

Lemma 4 ($\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ time-advantage). *$\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ achieves $(\Delta-\delta)$-time-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $\mathcal{T}_i^t \preceq \mathcal{T}_j^{t+\Delta-\delta}$.*

Proof. By Lemma 2, at the end of every epoch e, i.e., $t = e \cdot \Delta$, for any two participants (p_i,p_j), $\mathcal{T}_i^t = \mathcal{T}_j^t$. Starting from time t,the adversary $\mathcal{A}$ will learn the next random output no earlier than $t+\delta$, while correct participants will learn the next random output no later than $t+\Delta$, leading to $(\Delta-\delta)$-time-advantage.

Theorem 3 ($\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ delivery-fairness). *$\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ achieves (ω,ψ)-delivery-fairness where $\omega=1$ and $\psi=\Delta-\delta$.*

Proof. By Lemma 3-4, this theorem holds.

B.2 HydRand

Specification. Figure 8 outlines the specification of Π_{HydRand} and $\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ from the perspective of participant p_i in epoch e.

Delivery-fairness of non-lock-step HydRand. We provide the formal delivery-fairness analysis for non-lock-step Drand Π_{HydRand} below.

Lemma 5 (Π_{HydRand} length-advantage). *Π_{HydRand} achieves $(\frac{\Delta}{3\delta}+f)$-length-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $(\mathcal{T}_i^t)^{\lceil \frac{\Delta}{3\delta}+f} \preceq \mathcal{T}_j^t$.*

Proof. Recall that Π_{HydRand} has 3δ best-case and 3Δ worst-case latency. After a Δ of the latency manipulation attack, $\mathcal{A}$ and $f+1$ correct participants learn at most $\frac{\Delta}{3\delta}+1$ random outputs when all leaders during this Δ belong to them. Meanwhile, the rest correct participant only knows the first random output. This gives $\mathcal{A}$ the length-advantage of $\frac{\Delta}{3\delta}$. In addition, $\mathcal{A}$ can launch the private beacon attack over its latest block in order to obtains additional length-advantage of c, which is at most f. In total, $\mathcal{A}$ obtains the length-advantage degree $\omega=\frac{\Delta}{3\delta}+f$.

Lemma 6 (Π_{HydRand} time-advantage). *Π_{HydRand} achieves $((3f+1)\Delta-\delta)$-time-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $\mathcal{T}_i^t \preceq \mathcal{T}_j^{t+(3f+1)\Delta-\delta}$.*

Proof. Similar to analysis of Π_{Drand}, $\mathcal{A}$ can obtain $\Delta-\delta$ time-advantage in Π_{HydRand} via the latency manipulation attack. Since generating a random output takes at most 3Δ, $\mathcal{A}$ can obtain $3c\Delta$ time-advantage via the private beacon attack, where c is at most f. In total, $\mathcal{A}$ obtains time-advantage degree $\psi=(3f+1)\Delta-\delta$.

Theorem 4 (Π_{HydRand} delivery-fairness). *Π_{HydRand} achieves (ω,ψ)-delivery-fairness where $\omega=\frac{\Delta}{3\delta}+f$ and $\psi=(3f+1)\Delta-\delta$.*

Proof. By Lemma 5-6, this theorem holds.

Delivery-fairness of lock-step HydRand. We provide the formal delivery-fairness analysis for lock-step Drand $\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ below.

Lemma 7 ($\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ length-advantage). *$\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ achieves $(f+1)$-length-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $(\mathcal{T}_i^t)^{\lceil f+1} \preceq \mathcal{T}_j^t$.*

Proof. Assuming $\mathcal{A}$ does not corrupt the leader l_e in the current epoch e. In epoch e, $\mathcal{A}$ can launch the latency manipulation attack, so that it learns R_e earlier than correct participants. In addition, $\mathcal{A}$ can corrupt c future leaders $l_{e+1},...,l_{e+c}$ and learn further c random outputs $R_{e+1},...,R_{e+c}$ with probability $\Pr[X(\Pi_{\mathsf{LE}},n,f,c)]$, where c is at most f. Therefore, $\mathcal{A}$ achieves $(f+1)$-length-advantage.

Lemma 8 ($\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ time-advantage). *$\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ achieves $((3f+1)\Delta-\delta)$-time-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $\mathcal{T}_i^t \preceq \mathcal{T}_j^{t+(3f+1)\Delta-\delta}$.*

Proof. Assuming when epoch e starts at time t, $\mathcal{A}$ does not corrupt the leader l_e. $\mathcal{A}$ directs corrupted participants to follow the protocol while launching the latency manipulation attack, so that after three consecutive message transfers, $\mathcal{A}$ learns R_e at time $t+3\delta$ and the other correct participants learn R_e at time $t+\Delta+2\delta$. In addition, the leader can corrupt further c consecutive leaders $l_{e+1},\ldots,l_{e+c}$ and learns further c random outputs $R_{e+1},\ldots,R_{e+c}$ with probability $\Pr[X(\Pi_{\mathsf{LE}},n,f,c)]$, where c is at most f. Meanwhile, the correct participants learn R_{e+c} only at $t+\Delta+2\delta+3c\cdot\Delta$. Therefore, the time advantage is $(t+\Delta+2\delta+3f\cdot\Delta)-(t+3\delta)=(3f+1)\Delta-\delta$.

Theorem 5 ($\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ delivery-fairness). *$\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ achieves (ω,ψ)-delivery-fairness where $\omega=f+1$ and $\psi=(3f+1)\Delta-\delta$.*

Proof. By Lemma 7-8, this theorem holds.

B.3 GRandPiper

Specification. Figure 9 outlines the specification of $\Pi_{\mathsf{GRandPiper}}$ from the perspective of participant p_i in epoch e.

Delivery-fairness of GRandPiper. We provide the formal delivery-fairness analysis for GRandPiper $\Pi_{\mathsf{GRandPiper}}$ below.

Lemma 9 ($\Pi_{\mathsf{GRandPiper}}$ length-advantage). *$\Pi_{\mathsf{GRandPiper}}$ achieves $(f+1)$-length-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $(\mathcal{T}_i^t)^{\lceil f+1} \preceq \mathcal{T}_j^t$.*

Proof. The proof is identical to that of Lemma 7.

Lemma 10 ($\Pi_{\mathsf{GRandPiper}}$ time-advantage). *$\Pi_{\mathsf{GRandPiper}}$ achieves $((11f+1)\Delta-\delta)$-time-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $\mathcal{T}_i^t \preceq \mathcal{T}_j^{(11f+1)\Delta-\delta}$.*

Proof. By Lemma 9, for every epoch e, i.e., at time $t = e \cdot t_{\mathsf{BB}} + \delta$, the adversary learns the $(e+c)$-th random output with probability $\Pr[X(\Pi_{\mathsf{LE}}, n, f, c)]$, where c is at most f. Meanwhile, correct participants will learn the $(e+f)$-th random output after c epochs plus a Δ in the DRB routine, i.e., at time $t=(e+f)\cdot t_{\mathsf{BB}}+\Delta$. This leads to $(f\cdot t_{\mathsf{BB}}+\Delta-\delta)$-time-advantage. As $t_{\mathsf{BB}}=11\Delta$, $f\cdot t_{\mathsf{BB}}+\Delta-\delta=(11f+1)\Delta-\delta$.

Theorem 6 ($\Pi_{\mathsf{GRandPiper}}$ delivery-fairness). *$\Pi_{\mathsf{GRandPiper}}$ achieves (ω,ψ)-delivery-fairness where $\omega=f+1$ and $\psi=(11f+1)\Delta-\delta$.*

Proof. By Lemma 9-10, this theorem holds.

B.4 SPURT

Specification. Figure 10 outlines the specification of Π_{SPURT} and $\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ from the perspective of participant p_i in epoch e.

Delivery-fairnes of lock-step SPURT. We then analyse the delivery-fairness guarantee of $\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$, and show that $\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ achieves the optimal $(1, \Delta-\delta)$-delivery-fairness.

Lemma 11 ($\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ epoch execution). *In $\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$, at the end of every epoch e, i.e., $t=e\cdot(3\Delta+t_{\mathsf{BB}})$, for any two participants (p_i,p_j), $\mathcal{T}_i^t=\mathcal{T}_j^t$.*

Proof. The proof is identical to that of Lemma 2.

Lemma 12 ($\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ length-advantage). *$\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ achieves 1-length-advantage. That is, for any κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $(\mathcal{T}_i^t)^{\lceil 1} \preceq \mathcal{T}_j^t$.*

Proof. The proof is identical to that of Lemma 9, except that the adversary does not learn c extra random outputs via the private beacon attack compared to correct participants.

Lemma 13 ($\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ time-advantage). *$\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ achieves $(\Delta-\delta)$-time-advantage. That is, for every κ, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds except for probability $\mathsf{negl}(\kappa)$. For any two participants (p_i,p_j) and any time t, $\mathcal{T}_i^t \preceq \mathcal{T}_j^{\Delta-\delta}$.*

Proof. The proof is identical to that of Lemma 10, except that the adversary does not learn c extra random outputs via the private beacon attack compared to correct participants.

Theorem 7 (**$\Pi^{\mathsf{LS}}_{\mathsf{SPURT}}$ delivery-fairness**). *$\Pi^{\mathsf{LS}}_{\mathsf{Drand}}$ achieves (ω,ψ)-delivery-fairness where $\omega=1$ and $\psi=\Delta-\delta$.*

Proof. By Lemma 12-13, this theorem holds.

1. **(Propose)** Upon a new random output R_{e-1}, participants execute as follows.
   (a) Participants execute Π_{LE} to determine the leader l_e.
   (b) Leader l_e chooses a new secret s, obtains shares, encrypted shares and share proofs $(\vec{\boldsymbol{s}}, \vec{\boldsymbol{c}}, \vec{\boldsymbol{\pi}}) \leftarrow \Pi_{\mathsf{PVSS}}.\mathsf{Share}(\{pk_i\}_{i\in[n]}, f+1, s)$, obtains the accumulation value z of $\vec{\boldsymbol{c}}$ via $z \leftarrow \Pi_{\mathsf{Acc}}.\mathsf{Eval}(k, \vec{\boldsymbol{c}})$, and obtains all witnesses $\vec{\boldsymbol{w}}$ via $w_i \leftarrow \Pi_{\mathsf{Acc}}.\mathsf{CreateWit}(k,z,c_i,\vec{\boldsymbol{c}})$ for every $i \in [n]$.
   (c) Leader l_e constructs a block B_e and broadcasts B_e. B_e includes 1) information of epoch e: $(e, \vec{\boldsymbol{c}}, \vec{\boldsymbol{\pi}}, \vec{\boldsymbol{w}}, z)$, 2) information of epoch e^-: $(e^-, s^-, H(B_{e^-}), CC(B_e^-))$, and 3) information of in-between epochs $k \in (e^-, e)$: $\{(R_k, RC(k))\}_{k\in(e^-,e)}$, where e^- is the last epoch where the leader honestly reveals its secret, $CC(B_{e^-})$ ia a collection of $\geq f+1$ signatures on B_{e^-}, R_k is the recovered secret and $RC(k)$ ia a collection of $\geq f+1$ signatures on B_k.
   (d) All participants set a timer Δ.
2. **(Acknowledge)** Upon receiving a valid block B_e {before/and} Δ expires, participant p_i executes as follows.
   (a) p_i broadcasts an `ACKNOWLEDGE` message containing $H(B_e)$ and $(e, R_e, s^-, B_{e^-}, H(B_{e^-}), \{R_k\}_{k\in(e^-,e)}, z)$ signed by l_e, and set a new timer Δ.
   (b) Otherwise, if Δ expires and no valid block B_e is received, participant p_i moves to the vote-recover phase and sets a new timer Δ.
3. **(Vote-confirm)** Upon receiving $\geq 2f+1$ `ACKNOWLEDGE` messages on a valid block B_e {before/and} Δ expires, participant p_i executes as follows.
   (a) p_i broadcasts a `CONFIRM` message containing $H(B_e)$, and sets a new timer Δ.
   (b) Upon receiving $\geq f+1$ `CONFIRM` messages {before/and} Δ expires (which is guaranteed), p_i commits B_e, calculates random output $R_e \leftarrow H(R_{e-1} \| s^-)$, and starts a new epoch.
   (c) Otherwise, if Δ expires and $< 2f+1$ `ACKNOWLEDGE` messages on a valid block B_e are received, p_i moves to the vote-recover phase and sets a new timer Δ.
4. **(Vote-recover)** If there exists a phase where the condition does not meet after the timer expires, participant p_i moves to the vote-recover phase to recover the secret s^- committed in B_e^- jointly with others. Specifically,
   (a) p_i obtains the decrypted share s_i^- via $\Pi_{\mathsf{PVSS}}.\mathsf{Decrypt}(sk_i, c_i^-)$ and broadcasts `RECOVER` message $(s_i^-, c_i^-, \pi_i^-, w_i^-, R_{e-1})$.
   (b) Upon receiving $\geq f+1$ `RECOVER` messages {before/and} Δ expires (which is guaranteed), p_i recovers the secret $s^- \leftarrow \Pi_{\mathsf{PVSS}}.\mathsf{Recon}(\overrightarrow{\boldsymbol{s^-}})$ (where $\overrightarrow{\boldsymbol{s^-}}$ is the $\geq f+1$ shares in `RECOVER` messages), and calculates random output $R_e \leftarrow H(R_{e-1} \| s^-)$.

Fig. 8: Specification of HydRand Π_{HydRand}. Extra specification of its lock-step variant $\Pi^{\mathsf{LS}}_{\mathsf{HydRand}}$ is labelled in blue.

1. **(SMR routine)** Upon $\mathsf{timer}_{e-1} = t_{\mathsf{BB}}$ ends, all participants execute as follows.
   (a) **Leader election.** Participants calculate the leader based on the round-robin approach same as HydRand. All participants set a new $\mathsf{timer}_e = t_{\mathsf{BB}}$.
   (b) **Block proposal.** If elected as leader, leader l_e chooses a random value s_e, executes $(\vec{\boldsymbol{s}}, \vec{\boldsymbol{c}}, \vec{\boldsymbol{\pi}}) \leftarrow \Pi_{\mathsf{PVSS}}.\mathsf{Share}(\{pk_i\}_{i\in[n]}, f+1, s)$, creates block $B_e = (\vec{\boldsymbol{c}}, \vec{\boldsymbol{\pi}})$, and triggers the Byzantine broadcast protocol Π_{BA} over B_e. Each Π_{BA} instance terminates in at most f epochs.
   (c) **Block agreement.** Upon agreeing on block B_e^- sent by leader l_{e^-} for epoch e^-, participant p_i pushes B_e^- into queue $Q(l_{e^-})$.
   (d) **Blame.** Upon timer_e ends, if no block is proposed in epoch $e-t$, then remove l_{e-t} from future proposals.
2. **(DRB routine)** Upon timer_{e-1} ends, participant p_i executes as follows.
   (a) p_i pops the committed block $B_{e^-} = (\overrightarrow{\boldsymbol{c}^-}, \overrightarrow{\boldsymbol{\pi}^-})$ from queue $Q(l_{e^-})$.
   (b) p_i decrypts its share $s_i^- \leftarrow \Pi_{\mathsf{PVSS}}.\mathsf{Decrypt}(sk_i, c_i^-)$ and broadcasts s_i^-.
   (c) Upon $f+1$ valid shares in $\overrightarrow{\boldsymbol{s}^-}$, Participant p_i reconstructs $s_{e^-} \leftarrow \Pi_{\mathsf{PVSS}}.\mathsf{Recon}(\overrightarrow{\boldsymbol{s}^-})$, and calculates the random output $R_{e^-} \leftarrow H(s_{e^-}, R_{e^--1}, ..., R_{e^--t})$.

Fig. 9: Specification of GRandPiper $\Pi_{\mathsf{GRandPiper}}$.

1. **(Commitment)** Upon reconstructing R_{e-1}, every participant p_i executes as follows.
   (a) p_i executes $(\vec{\boldsymbol{s_i}},\vec{\boldsymbol{c_i}},\vec{\boldsymbol{\pi_i}}) \leftarrow \Pi_{\mathsf{PVSS}}^{\mathsf{uniform}}.\mathsf{Share}(\{pk_j\}_{j\in[n]},f,s)$.
   (b) p_i sends tuple $(\vec{\boldsymbol{c_i}},\vec{\boldsymbol{\pi_i}})$ to leader l_e.
   (c) Set a timer Δ.
2. **(Aggregation)** Upon receiving $f+1$ tuples and Δ expires, leader l_e executes as follows.
   (a) l_e homomorphically aggregates them as $(\hat{\boldsymbol{c}}, \hat{\boldsymbol{\pi}}) = \{(\hat{c_i}, \hat{\pi_i})\}_{i\in[n]} = \{(\Pi\bar{\boldsymbol{c_i}},\Pi\bar{\boldsymbol{\pi}_i})\}_{i\in[n]}$, where $(\bar{\boldsymbol{c_i}},\bar{\boldsymbol{\pi}_i})$ is the i-th column of $f+1$ encrypted shares and proofs, respectively.
   (b) l_e computes the digest $h \leftarrow H(I\|\hat{\boldsymbol{c}})$ where I is the set of $f+1$ indices.
3. **(Agreement)** Leader l_e executes as follows.
   (a) For each participant p_i, l_e sends $(e,h,I,\hat{\boldsymbol{c}},\bar{\boldsymbol{c_i}},\bar{\boldsymbol{\pi}_i})$ to p_i.
   (b) l_e triggers Π_{BB} over h with all participants.
   (c) Set a timer t_{BB}.
4. **(Reconstruction)** Upon deciding on h and t_{BB} expires, participant p_i executes as follows.
   (a) p_i decrypts the aggregated share $\hat{s_i} \leftarrow \Pi_{\mathsf{PVSS}}^{\mathsf{uniform}}.\mathsf{Decrypt}(sk_i, \hat{c_i})$, and broadcasts $\hat{s_i}$. Set a timer Δ.
   (b) Upon receiving $f+1$ such decrypted shares $\hat{\boldsymbol{s}}$ and Δ expires, p_i homomorphically aggregates them to $s \leftarrow \sum \hat{\boldsymbol{s}}$, calculates the beacon output $R_e \leftarrow e(h_0^s,h_1)$ via pairing, and broadcasts R_e. Set a timer Δ.
   (c) Upon receiving $f+1$ R_e messages and Δ expires, p_i decides on R_e.

Fig. 10: Specification of SPURT Π_{SPURT}. Extra specification of its lock-step variant $\Pi_{\mathsf{SPURT}}^{\mathsf{LS}}$ is labelled in blue.