Efficient Asynchronous Byzantine Agreement without Private Setups

Yingzi Gao[1,4], Yuan Lu[1], Zhenliang Lu[2], Qiang Tang[3], Jing Xu[1], and Zhenfeng Zhang[1]

[1] Institute of Software Chinese Academy of Sciences
{yingzi2019,luyuan,xujing,zhenfeng}@iscas.ac.cn
[2] luzhenliang1992@gmail.com
[3] The University of Sydney
qiang.tang@sydney.edu.au
[4] University of Chinese Academy of Sciences

Abstract. Though recent breakthroughs have greatly improved the efficiency of asynchronous Byzantine agreement protocols, they mainly focused on the setting with private setups, e.g., assuming a trusted dealer to establish non-interactive threshold cryptosystems. Challenges remain to reduce the large communication complexities in the absence of private setups, for example: (i) for asynchronous binary agreement (ABA) with optimal resilience, prior private-setup free protocols (Cachin et al., CCS' 2002; Kokoris-Kogias et al., CCS' 2020) have to incur $\mathcal{O}(\lambda n^4)$ bits and $\mathcal{O}(n^3)$ messages; (ii) for asynchronous multi-valued agreement with external validity (VBA), Abraham et al. [2] very recently gave the first elegant construction with $\mathcal{O}(n^3)$ messages, relying on only public key infrastructure (PKI), but the design still costs $\mathcal{O}(\lambda n^3 \log n)$ bits. Here n is the number of participating parties and λ is the cryptographic security parameter.

We for the first time close the remaining efficiency gap between the communication complexity and the message complexity of private-setup free asynchronous Byzantine agreements, i.e., reducing their communication cost to only $\mathcal{O}(\lambda n^3)$ bits on average. At the core of our design, we give a systematic treatment of reasonably fair common randomness, and proceed as follows:

- We construct a reasonably fair common coin (Canetti and Rabin, STOC' 1993) in the asynchronous setting with PKI instead of private setup, using only $\mathcal{O}(\lambda n^3)$ bit and constant asynchronous rounds. The common coin protocol ensures that with at least 1/3 probability, all honest parties can output a common bit that is as if uniformly sampled, rendering a more efficient private-setup free ABA with expected $\mathcal{O}(\lambda n^3)$ bit communication and constant running time.
- More interestingly, we lift our reasonably fair common coin protocol to attain perfect agreement without incurring any extra factor in the asymptotic complexities, resulting in an efficient reasonably fair leader election primitive pluggable in all existing VBA protocols (including Cachin et al., CRYPTO' 2001; Abraham et al., PODC' 2019; Lu et al., PODC' 2020), thus reducing the communication of private-setup free VBA to expected $\mathcal{O}(\lambda n^3)$ bits while preserving expected constant running time. This leader election primitive and its construction might be of independent interest.
- Along the way, we also improve an important building block, asynchronous verifiable secret sharing (Canetti and Rabin, STOC' 1993) by presenting a private-setup free implementation costing only $\mathcal{O}(\lambda n^2)$ bits in the PKI setting. By contrast, prior art having the same communication complexity (Backes et al., CT-RSA' 2013) has to rely on a private setup.

Keywords: asynchronous byzantine agreement · asynchronous verifiable secret sharing · common coin · leader election

1 Introduction

Recently, following the unprecedented demand of deploying BFT protocols on the Internet for robust and highly available decentralized applications, renewed attentions are gathered to implement more efficient asynchronous Byzantine agreements [3,24,29,21,28]. Nevertheless, asynchronous protocols have to rely on randomized executions to circumvent the seminal FLP "impossibility" result [20]. In particular, to quickly decide the output in expected constant time, many asynchronous protocols [15,30,29,13,12,3,28,21,24,17,9] essentially need *common* randomness, given which for free, one can construct optimally resilient asynchronous Byzantine agreements that are at least as efficient as using expected $\mathcal{O}(n^2)$ messages and constant running time [3,28,30,13].

However, efficient ways to implement asynchronous common randomness in practice mostly rely on different varieties of private setups. For example, initialed by M. Rabin [32], it can directly assume that a trusted dealer uses a secret sharing scheme to distribute a large number of secrets among the participating parties before the protocol execution, which later can be collectively reconstructed by the parties to generate a sequence of random bits. Later, Cachin et al. [13] presented how to construct a non-interactive threshold pseudorandom function (tPRF) assuming that a trusted dealer can faithfully share a short tPRF key, which now is widely used by most practical asynchronous BFT protocols including [3,24,29,28].

These private setups might cause unpleasant deployment hurdles, preventing asynchronous BFT from being widely used in a broader array of applications. Hence, it becomes critical to reduce the setup assumptions for easier real-world deployment.

Existing efforts on reducing setups. There are a few known approaches to construct private-setup free asynchronous Byzantine agreements, but most are costly or even prohibitively expensive.

Back to 1993, Canetti and Rabin [15] gave a beautiful common coin construction (CR93) centering around asynchronous verifiable secret sharing (AVSS), from which a fast and optimally resilient asynchronous binary agreement (ABA) can be realized. Here, AVSS is a two-phase protocol that allows a dealer to confidentially "commit" a secret across n participating parties during a sharing phase, after which a reconstructing phase can be successively executed to make all honest parties recover the committed secret. The resulting *common coin* is *reasonably fair*, as it ensures all honest parties to output either 0 or 1 with some constant probability. Though this reasonably fair common coin attains constant asynchronous rounds and can be directly plugged into many binary agreement constructions [30,15], it incurs tremendous $\mathcal{O}(n^6)$ messages and $\mathcal{O}(\lambda n^8 \log n)$ bits (where λ is the security parameter). The huge complexities are dominated by its expensive AVSS, and many more efficient private-setup free AVSS protocols [11,7,4,8] can improve it. For example, Cachin et al. [11] gave an AVSS to share n secrets with only $\mathcal{O}(n^2)$ messages and $\mathcal{O}(\lambda n^3)$ bits, but the resulting common coin and ABA protocols (CKLS02) would still incur $\mathcal{O}(n^3)$ messages and $\mathcal{O}(\lambda n^4)$ bits, which remains expensive and exists a $\mathcal{O}(n)$ gap between the message and the communication complexities.

Recently, Kokoris-Kogias et al. [25] presented a new path to reducing common coin to AVSS, which can save the amortized communications but slightly trade off the running time. To generate a single random coin, Kokoris-Kogias et al.'s protocol (KMS20) incurs $\mathcal{O}(\lambda n^4)$ bits and $\mathcal{O}(n)$ asynchronous rounds, which is seemingly worse than CKLS02; but, it has better amortized complexities, because it can generate perfect fair random coins at a cost of constant rounds and $\mathcal{O}(\lambda n^2)$ bits per each, after executing at most $\mathcal{O}(n)$ rounds, so it can be cheaper in an amortized way. Noticeably, the design requires a strengthened AVSS notion with high-threshold secrecy, which specifies that the adversary shall not learn the secret "committed" during the sharing phase, unless $n-2f$ honest parties tried to reconstruct the secret (where f is the number of probably corrupted parties).[1]

In a very recent breakthrough, Abraham et al. [2] presented an elegant (validated) agreement protocol that avoids private setups to tolerate asynchronous network and maximal $n/3$ Byzantine corruptions in the presence of a bulletin PKI that only aids in the registration and broadcast of public keys. For λn-bit input, it spends $\mathcal{O}(\lambda n^3 \log n)$ bits, $\mathcal{O}(n^3)$ messages, and expected constant running time. It also defines and constructs a novel proposal election primitive, which attains a reasonably constant probability that the honest parties can elect a common value proposed by some honest party, such that a certain protocol called *No-Waitin' HotStuff* can be accordingly tailored in the asynchronous network to realize the validated Byzantine agreement (VBA). Nevertheless, this private-setup free VBA (AJM+21) has an $\mathcal{O}(\log n)$ gap between the message and communication complexities, leaving a room to further improve the asymptotic communication complexity. In addition, the proposal election primitive cannot be directly plugged in most recent efficient VBA constructions [3,28,12].

[1] Different from high-threshold AVSS, classic AVSS in CR93 [15] only preserves secrecy before the first honest party activates reconstruction. Moreover, classic AVSS is even weaker in another aspect, as it might only ensure $f+1$ honest parties to receive their corresponding shares. Instead, many advanced notions [25,11,4,7] require strong commitment property that implies all honest parties to receive the corresponding shares.

Bearing the start-of-the-art of private-setup free asynchronous Byzantine agreements, it calls for more general approaches to systematically treat asynchronous protocols with relying on fewer setup assumptions, e.g., more general (reasonably fair) common randomness notions that are pluggable in most existing Byzantine agreement protocols as well as efficiently implementable without private setups, and thus the following question remains open:

Can we give a systematic treatment of private-setup free asynchronous Byzantine agreements (e.g., ABA *and* VBA*) to further improve their efficiency, e.g., attain expected $\mathcal{O}(\lambda n^3)$-bit communication, thus closing the existing efficiency gaps between their message and communication complexities?*

1.1 Our contribution

We give an affirmative answer to the above question. At the core of our solution, we present a new efficient private-setup free construction for reasonably fair common coin that are pluggable in many existing ABA protocols [15,30,18]; more interestingly, we formalize and construct an efficient (reasonably fair) leader election with perfect agreement such that it can be directly plugged in all existing VBA protocols [12,3,28] to remove private setup. In greater detail, our technical contribution is three-fold:

- We give an AVSS construction satisfying the classic CR93 notion [15] with only bulletin PKI (and discrete logarithm assumption), and it costs only $\mathcal{O}(n^2)$ messages and $\mathcal{O}(\lambda n^2)$ bits when sharing λ-bit secret. To our knowledge, this is the first private-setup free AVSS that attains $\mathcal{O}(\lambda n^2)$ communication complexity, and prior art either relies on private setup [6,23] or incurs at least $\mathcal{O}(\lambda n^3)$ bits [11,7] (except a very recent work [4], yet it still has an extra $\log n$ factor than ours).
- We implement private-setup free ABAs with expected $\mathcal{O}(n^3)$ message complexity and $\mathcal{O}(\lambda n^3)$ communication complexity with only bulletin PKI. As illustrated in Table 1, it closes the $\mathcal{O}(n)$ gap between the message and the communication complexities in the earlier private-setup free ABA protocols such as CKLS02 [11], while preserving other benefits such as the maximal $n/3$ resilience and the optimal expected constant running time. Even comparing with a very recent work due to Abraham et al. [2] that presents a more efficient VBA construction and improves ABA as a by-product,[2] our approach still realizes a $\log n$ factor improvement.

  The crux of our design is a novel efficient construction for the reasonably fair common coin in the bulletin PKI setting (conditioned on the random oracle model), with using the more efficient AVSS protocol and verifiable random function. This private-setup free common coin costs only $\mathcal{O}(\lambda n^3)$ bits and constant asynchronous rounds.
- We further present how to efficiently instantiate private-setup free VBAs (i.e., multi-valued Byzantine agreement with external validity) in the asynchronous setting. For λn-bit input, the resulting VBA realizes the maximal $n/3$ resilience and optimal expected constant running time, with costing expected $\mathcal{O}(n^3)$ messages and $\mathcal{O}(\lambda n^3)$ bits. As shown in Table 1, this construction closes the $\mathcal{O}(\log n)$ gap between the message and the communication complexities of VBA protocols. In addition, as a by-product, our VBA construction can be directly plugged in the asynchronous distributed key generation protocol in AJM+21 and reduces its communication cost by an $\mathcal{O}(\log n)$ order to realize $\mathcal{O}(\lambda n^3)$ communication complexity.

  To implement more efficient VBA without private setup, we construct a leader election primitive with reasonable fairness and perfect agreement without private setups, assuming the random oracle model. The design costs only $\mathcal{O}(\lambda n^3)$ bits and expected constant asynchronous rounds, and can directly be plugged in all existing VBA protocols [12,3,28] to replace its counterpart relying on private setups, which can be of independent interests.

[2] Remark that there might exist complexity-preserving reduction from ABA to VBA, which was discussed in [12]. The idea is simple: every party signs and multicasts its input bit, then each one solicits a vector of $n-f$ input-signature pairs from distinct parties, and feeds the vector into VBA, such that VBA would return to everyone the common vector of $n-f$ signed bits, the majority of which becomes the ABA output. Therefore, the very recent VBA protocol in [2] also improves ABA in the setting with only public key infrastructure.

Table 1. Complexities of private-setup free asynchronous protocols with optimal resilience. Message complexity is omitted, as all one-shot protocols in the Table costs $\mathcal{O}(n^3)$ messages, except that n ABAs cost $\mathcal{O}(n^4)$ messages.

Protocols w/o private setup	**Earlier Results**			**Our Result**	
	Results	Communication Complexity	Running Time	Communication Complexity	Running Time
(Reasonably fair) common coin	CKLS02 [11]	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(1)$	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$
	KMS20 [25]	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(n)$		
	AJM+21 [2]†	$\mathcal{O}(\lambda n^3 \log n)$	$\mathcal{O}(1)$		
(Reasonably fair) leader election with perfect agreement	AJM+21 [2]†	$\mathcal{O}(\lambda n^3 \log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$
Binary agreement (ABA)	CKLS02 [11]	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(1)$	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$
	AJM+21 [2]‡	$\mathcal{O}(\lambda n^3 \log n)$	$\mathcal{O}(1)$		
n ABAs instances in parallel	CKLS02 [11]	$\mathcal{O}(\lambda n^5)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(\log n)$
	KMS20 [25]	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(n)$		
	AJM+21 [2]‡	$\mathcal{O}(\lambda n^4 \log n)$	$\mathcal{O}(\log n)$		
Validated agreement (VBA)	AJM+21 [2]	$\mathcal{O}(\lambda n^3 \log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$
Asynchronous DKG	KMS20 [25]	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$
	AJM+21 [2]	$\mathcal{O}(\lambda n^3 \log n)$	$\mathcal{O}(1)$		

† Note that AJM+21 [2] did not present any explicit constructions for the reasonably fair common coin and leader election. Nevertheless, this very recent work gave an asynchronous distributed key generation protocol with $\mathcal{O}(\lambda n^3 \log n)$ bits and expected constant running time, which can potentially bootstrap threshold verifiable random function and can faithfully set up common coin and leader election schemes. However, this unnecessarily long path to constructing common randomness protocols is essentially improvable to our results.

‡ Note that AJM+21 [2] did not explicitly give any constructions for asynchronous binary agreement (ABA). Nonetheless, there is a simple complexity-preserving reduction from ABA to VBA in the PKI setting [12].

1.2 Technical overview

Either CR93 or AJM+21 has to require every party to verifiably share or reliably broadcast $\mathcal{O}(\lambda n)$ bits in the asynchronous network. In CR93, this is because every party needs to play a role of "delegate" for each party to share a secret through AVSS, so everyone can fix $n-f$ secrets shared by distinct "delegates" before learning all these secrets, such that the aggregation of the $n-f$ secrets cannot be biased. In AJM+21, each party also play a role of "delegate" for everyone to generate a secret confidentially committed to an aggregatable public verifiable secret sharing (PVSS) script, so everyone can combine $n-f$ secrets hidden behind $n-f$ PVSS scripts from distinct "delegates" to obtain an aggregated PVSS script, which has λn bits and hides a secret. The aggregated PVSS script then needs to be sent to all parties via reliable broadcast (analog to the sharing phase of AVSS, though there is no explicit AVSS invocation), thus bootstrapping the setup for threshold verifiable random function (tVRF) and enabling a proposal election primitive. As such, further reducing the communication of the CR93 and AJM+21 frameworks seems challenging because every party has to reliably broadcast (or share) $\mathcal{O}(\lambda n)$ bits. On the other side, KMS20 lifts AVSS to high-threshold AVSS and reduces the number of secrets to share by an $\mathcal{O}(n)$ order, but still needs $\mathcal{O}(\lambda n^4)$ bits in the worst case, and even incurs a running time of $\mathcal{O}(n)$ rounds.

We pave a novel path to closing the gap between the message and communication complexities of asynchronous Byzantine agreement protocols without private setups in the bulletin PKI setting. As shown in Fig. 1, our new method can be described as follows:

Reducing the number of bits to share in Common Coin. We firstly realize that if one would use aggregatable PVSS to generate an unpredictable nonce instead of establishing tVRF, it is no longer necessary to broadcast the PVSS script aggregating $n-f$ parties' contributions to the whole network. For the purpose, we set forth a notion called led reliable seeding (Seeding) and construct it, in which a leader can follow the pattern of Reiter's consistent broadcast [34] to send the same PVSS script to at least $n-f$ parties; in addition, a party receives the PVSS script can immediately decrypt the script to get its share and return it to the leader, such that the leader can recover the actual secret hidden

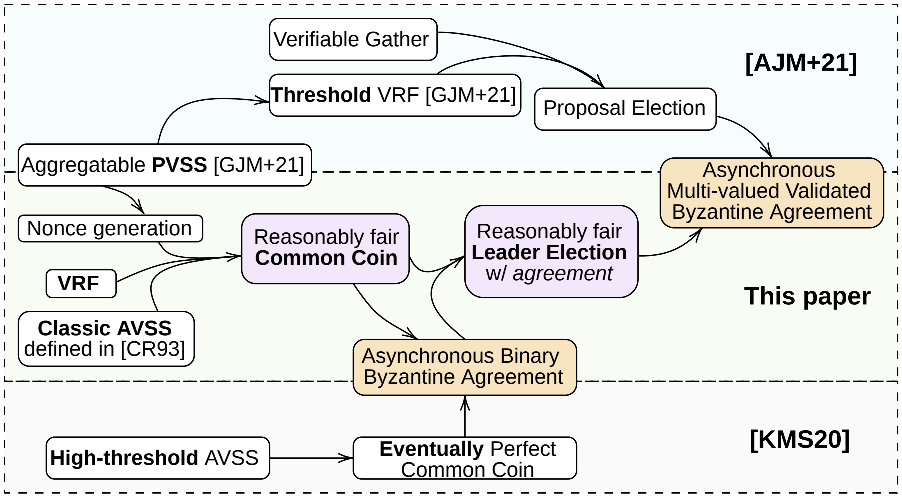

Fig. 1. Our technique in comparison with some recent related results (e.g., KMS20 [25] and AJM+21 [2]). The arrows roughly depict the path of constructions instead of the precise relationship of reducing.

behind PVSS, and follows the pattern of Bracha's reliable broadcast [10] to broadcast the recovered secret (also called seed or nonce by us); moreover, at least $n-2f$ honest parties can verify whether the recovered seed is the one committed to PVSS or not, and then decide whether to participate into the broadcast, so if any honest party outputs the seed, then all honest parties must receive the same seed committed to the same PVSS script.

Then, we observe that one can augment its verifiable random function (VRF) by using the Seeding output as VRF seed, because the seed is always unpredictable until it is fixed by committing to the PVSS script signed by $n-f$ parties. Now, no party can bias its own VRF evaluation result, because such VRF seed can remain unpredictable (thus cannot be used to query VRF) until committed. This actually hints to us that a single VRF evaluation already can replace the functionality of n secrets in the CR93 framework, as CR93 uses $\mathcal{O}(n)$ secrets shared by distinct parties towards combining a single random value on behalf of each probably malicious party. So our new Coin construction starts by letting each party use AVSS to share its own VRF evaluation-proof pair; then, until at least $n-f$ parties receive the outputs from a set of AVSS instances covering the same $n-f$ AVSSes, i.e., a core set called by literature [15,2], but our core set is slightly weaker, because probably only $f+1$ honest parties (instead of all honest parties) share the same core. This weaker core set can be easily obtained in the PKI setting without traditional three-phase reliable broadcasts [1,5], and its existence can be verified by a quorum certificate proof (because we are in the PKI setting). After fixing a quorum proof for any core set, the parties then start reconstruct all AVSS instances to reveal all hidden VRF evaluations, and use the most significant bits of the VRF evaluations to pick the largest VRF, the lowest bit of which naturally becomes the final output bit.

As such, we only need each party to share $\mathcal{O}(\lambda)$-bit secret in our new common coin framework, and more importantly, with at least $1/3$ probability, our seemingly weaker core set can solicit the largest VRF evaluated by an honest party, such that at least $f+1$ honest parties can obtain this common largest VRF evaluation, and then they can multicast the largest VRF to guarantee all honest parties to see it. So with probability $1/3$, a good event happens that all honest parties have the common largest VRF in their views, which is also evaluated by some honest party. In addition, an honest party's VRF evaluation is unpredictable by the adversary before the core set is fixed and the AVSS reconstructions start, such that it ensures all parties to obtain a common unpredictable bit with at least $1/3$ chance.

Minimizing the communication of Classic AVSS. Meanwhile, in order to reduce the number of bits costed by the n AVSS instances in our Coin construction, we for the first time design an AVSS protocol attaining $\mathcal{O}(\lambda n^2)$ communication complexity with only the bulletin PKI assumption. Noticing that the classic AVSS notion in CR93 (e.g., without the fancy high-threshold secrecy) is already enough in our new Coin framework, we lift Pedersen's verifiable secret sharing [31] in the asynchronous network, with exploiting the wisdom of hybrid secret sharing [26] to split the AVSS construction into two stages: the first stage follows Reiter's consistent broadcast [34] to make at least $f+1$ parties agree on the same polynomial commitment to bind a unique encryption key, and the second stage roughly follows Bracha's reliable broadcast [10] to send a ciphertext encrypting the actual secret to all parties, which minimizes the communication by avoid broadcasting the large polynomial commitment to the whole network. An extra challenge might happen in the reconstruction phase as probably only $f+1$ honest parties receive the polynomial commitment and can recover the encryption key during the sharing phase; nevertheless, we let these parties to broadcast the recovered key such that all parties can eventually receive the same key to decrypt their ciphertext.

Realizing perfect agreement for reasonable Leader Election. Although the common coin primitive can be a powerful tool to enable Binary agreement in the asynchronous network, it is facing a few barriers for implementing the interesting class of validated Byzantine agreements (VBA), because most existing VBA constructions [12,3,28] require the underlying leader election primitive to have at least two strengthenings relative to common coin: first, there shall be a reasonable constant probability that all parties' outputs are as if being uniformly chosen from the indices of all parties instead of from only 0 and 1; second, all parties' outputs need to be consistent, otherwise, the agreement property in [12,3,28] can be directly broken.

Fortunately, the first strengthening can be straightly obtained by our Coin construction. To further lift the Coin protocol for perfect agreement, we design a set of specific voting rules: if everyone reliably broadcasts the speculative largest VRF heard at the end of Coin execution and then waits for $n-f$ such broadcasted VRF, they can vote 1 or 0 into a ABA protocol, according to whether there exists a majority VRF that is also the largest one among the $n-f$ received VRF evaluations. These voting rules can ensure that when ABA returns 1, at least one honest party can announce its $n-f$ received VRF evaluations to convince all honest parties to choose the same VRF as output, and no malicious party can make any two honest parties to accept two distinct VRF evaluations (because the VRF evaluation satisfying the voting rules must be unique), thus fixing the lack of agreement in common coin without incurring extra factors in complexities.

1.3 Other related work

Kokoris-Kogias et al. [25] and Abraham et al. [2] also lifted their private-setup free Byzantine agreement protocols to realize asynchronous distributed key generation (ADKG), thus emulating the trusted dealer of threshold cryptosystems. The rationale behind ADKG is that it might amortize the cost of asynchronous protocols, if the threshold cryptosystem can be used for many times. Notwithstanding, the idea would *not* save, if running one-shot agreement or having dynamic participants. In addition, more efficient ADKG also relies on more efficient private-setup free asynchronous Byzantine agreement, for example, our new path to constructing more efficient VBA naturally reduces the communication cost of AJM+21 ADKG by an $\log n$ order, thus realizing the fist ADKG protocol with expected constant running time and $\mathcal{O}(\lambda n^3)$ communication.

Recently, [17] gave a new asynchronous common coin protocol with sub-optimal resilience using VRFs, but the design uses pre-determined nonce for verifiable random functions (VRFs), so it actually takes an implicit assumption that a trusted third-party performs honest VRF key generations on behalf of all parties, otherwise, malicious parties can easily run a polynomial times of VRF key generations to choose the most preferable VRF keys and break their design.

There are many AVSS protocols [6,23] that can realize optimal communication with relying on private setups. Without private setup, the best known result is a very recent study [4] that incurs $\mathcal{O}(n^2)$ messages and $\mathcal{O}(\lambda n^2 \log n)$ bits for λ-bit input secret. Some AVSS protocols [23,4] also focus on linear amortized communication for sufficiently large input secret, but they still have to exchange quadratic messages and bits while sharing a short secret. Our AVSS can easily combine the information dispersal technique [14] to realize the same linear amortized communication.

2 Models

Fully asynchronous system without private setup. There are n designated parties, each of which has a unique identity (i.e., $\mathcal{P}_1$ through $\mathcal{P}_n$) known by everyone. Moreover, we consider the asynchronous message-passing model with static corruptions and bulletin public key infrastructure (PKI) assumption in the absence of any private setup. In particular, our system and threat models can be detailed as:

- *Bulletin PKI.* There exists a PKI functionality that can be viewed as a bulletin board, such that each party $\mathcal{P}_i \in \{\mathcal{P}_j\}_{j\in[n]}$ can register some public keys (e.g., the verification key of digital signature) bounded to its identity via the PKI before the start of protocol.
- *Computing model.* We let the n parties and the adversary $\mathcal{A}$ be probabilistic polynomial-time interactive Turing machines (ITMs). A party $\mathcal{P}_i$ is an ITM defined by the given protocol: it is activated upon receiving an incoming message to carry out some polynomial steps of computations, update its states, possibly generate some outgoing messages, and wait for the next activation. Moreover, we explicitly require the bits of the messages generated by honest parties to be probabilistic uniformly bounded by a polynomial in the security parameter λ, which naturally rules out infinite protocol executions and thus restrict the running time of the adversary through the entire protocol.
- *Up to $n/3$ static Byzantine corruptions.* The adversary can choose up to f out of n parties to corrupt and fully control, before the course of a protocol execution. No asynchronous BFT can tolerate more than $f = \lfloor (n-1)/3 \rfloor$ such static corruptions. Through the paper, we stick with this optimal resilience. We also consider that the adversary can control the corrupted parties to generate their key materials maliciously, which captures that the compromised parties might exploit advantages whiling registering public keys at the PKI.
- *Fully asynchronous network.* We assume that there exists an established p2p channel between any two parties. The channels are considered as *secure*, which means the adversary cannot modify or drop the messages sent between honest parties and cannot learn any information of the messages except their lengths. Moreover, the adversary must be consulted to approve the delivery of messages, namely, it can arbitrarily delay and reorder messages. Remark that we assume asynchronous secure channels (instead of merely asynchronous reliable channels) for presentation simplicity, and they are not extra assumptions as can be obtained from the PKI assumption.
- *Miscellany.* All system parameters, such as n, are (probably unfixed) polynomials in the security parameter λ [12,3,28].

Quantitative performance metrics. Since we are particularly interested in constructing efficient asynchronous protocols, e.g., for generating common randomness or reaching consensus, without private setup, it becomes needed to introduce quantitative metrics to define the term "efficiency" in the context. To this end, we consider the following widely adopted notions to quantify the performance of protocols in the asynchronous network:

- *Communication complexity* is defined as the bits of all messages exchanged among honest parties during a protocol execution. Sometimes, an asynchronous protocol might have randomized executions, so we might consider the upper bound of expected communication complexity (averaged over all possible executions) in the presence of any adversary that fully controls the network and corrupts up to $n/3$ parties.
- *Message complexity* captures the number of messages exchanged among honest parties in a protocol execution. Similar to communication's, we sometimes might consider the upper bound of expected message complexity.
- *Asynchronous rounds (or running time).* The eventual delivery of the asynchronous network notion might cause that the protocol execution is somehow independent to "real time". Nevertheless, it is needed to characterize the running time of asynchronous protocols. A standard way to do so is: for each message the adversary assigns a virtual round number r, subject to the condition that any $(r-1)$-th round messages between any two correct parties must be delivered before any $(r+1)$-th round message is sent [15]. So it becomes straightforward to measure the running time by counting such asynchronous "rounds".

Other notations. The cryptographic security parameter is denoted by λ through the paper, which captures the bit-length of signatures and hash digests. We let $\langle x, y\rangle$ denote a string concatenating two strings x and y. Any protocol message sent between two parties would be in the form of $\textsc{MsgType}(\mathsf{ID}, \cdot, \cdot, \cdots)$, where $\textsc{MsgType}$ specifies the message type and the ID field represents the protocol instance that the message belongs to. Our protocol description follows the conventional pseudocode notations to describe asynchronous protocols [12,3,28]: we let "**wait** for X do $\cdots$" to represent that the protocol is blocking until a certain event X happens to proceed; sometimes, we omit an implicit "**wait** for" handler for the incoming messages and use "**upon** receiving message Y do $\cdots$" to describe how to process an applicable incoming message Y. Moreover, we let $\Pi[\mathsf{ID}]$ refer to an instance of some protocol Π with an explicit identifier ID, and $y \leftarrow \Pi[\mathsf{ID}](x)$ means to invoke $\Pi[\mathsf{ID}]$ on input x and then block to wait for the output y before proceeding.

3 Preliminaries

Cryptographic hash function. A hash function $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\lambda$ is said to be collision-resistant, if no probabilistic polynomial-time $\mathcal{A}$ can generate two distinct values x_1 and x_2 s.t. $\mathcal{H}(x_1) = \mathcal{H}(x_2)$ except with negligible probability.

Reliable broadcast (RBC) is a protocol running among a set of n parties in which there is a party called sender whose aim is to broadcast a value to all the other parties. Formally, an RBC protocol satisfies the following properties:

- Agreement. If any two honest parties output v and v' respectively, then $v = v'$.
- Totality. If an honest party outputs v, then all honest parties output v.
- Validity. If the sender is honest and inputs v, then all honest parties output v.

Digital signature. A digital signature scheme consists of a tuple of algorithms ($\mathsf{KenGen}, \mathsf{Sign}, \mathsf{SigVerify}$):

- $\mathsf{KenGen}(1^\lambda) \rightarrow (sk, pk)$ is a probabilistic algorithm generating the signing and verification keys.
- $\mathsf{Sign}(sk, m) \rightarrow \sigma$ takes a signing key sk and a message m as input to compute a signature σ.
- $\mathsf{SigVerify}(pk, m, \sigma) \rightarrow 0/1$ verifies whether σ is a valid signature produced by a certain party with verification key pk for the message m or not.

The signature scheme shall satisfy *correctness* and *security*: the *correctness* requirement is trivial, i.e., for any signing-verification key pair, the honestly generated signature for any message shall pass the verification; for *security*, we require the digital signature scheme to be existentially unforgeable under an adaptive chosen-message attack (i.e., $\mathsf{EUF\text{-}CMA}$ secure).

In the bulletin PKI setting, every party is bounded to a unique verification key for digital signature scheme. For presentation brevity, in a protocol instance with an explicit identifier ID, we might let $\mathsf{Sign}_i^{\mathsf{ID}}(m)$ to denote $\mathsf{Sign}(sk_i, \langle \mathsf{ID}, m \rangle)$, which means a specific party $\mathcal{P}_i$ signs a message m with using its private key in the protocol instance, and also let $\mathsf{SigVerify}_i^{\mathsf{ID}}(m, \sigma)$ to denote $\mathsf{SigVerify}(pk_i, \langle \mathsf{ID}, m \rangle, \sigma)$, where pk_i is the public key of a specific party $\mathcal{P}_i$.

Verifiable random function. A verifiable random function is a pseudorandom function returning proofs for its correct computation. It consists of three algorithms ($\mathsf{VRF.Gen}, \mathsf{VRF.Eval}, \mathsf{VRF.Verify}$):

- $\mathsf{VRF.Gen}(1^\lambda) \rightarrow (sk, pk)$ is a probabilistic algorithm that generates a pair of private key and public verification key for verifiable random function.
- $\mathsf{VRF.Eval}(sk, x) \rightarrow (r, \pi)$ takes a secret key sk and a value x as input and outputs a pseudorandom value r with a proof π.
- $\mathsf{VRF.Verify}(pk, x, r, \pi) \rightarrow 0/1$ verifies whether r is correctly computed from x and sk using π and the corresponding pk.

The verifiable random function shall satisfy *unpredictability*, *verifiability* and *uniqueness*. The *verifiability* property means $\Pr[\ \mathsf{VRF.Verify}(pk, x, \mathsf{VRF.Eval}(sk, x)) = 1 \mid (sk, pk) \leftarrow \mathsf{VRF.Gen}(1^\lambda)\ ] = 1$. *Unpredictability* requires that for any input x, it is infeasible to distinguish the value $r = \mathsf{VRF.Eval}(sk, x)$ from another uniformly sampled value r' without access to sk. *Uniqueness* requires that it is infeasible to find x, r_1, r_2, π_1, π_2 such that $r_1 \neq r_2$ but $\mathsf{VRF.Verify}(pk, x, r_1, \pi_1) = \mathsf{VRF.Verify}(pk, x, r_2, \pi_2) = 1$. Due to our bulletin PKI assumption, everyone is associated to a unique VRF public key: an honest party $\mathcal{P}_i$ runs the $\mathsf{VRF.Gen}$ algorithm correctly to generate a unique pair of private key sk_i and public key pk_i, while a corrupt might exploit malicious key generate in order to bias its VRF distribution. To capture the threat of malicious key generations of VRF in the bulletin PKI model, we actually require a stronger *unpredictability* property called *unpredictability under malicious key generation* due to David et al. [19] through the paper, which means that even if the adversary is allowed to corrupt some parties to conduct malicious key generation, VRF remains to perform like a random oracle. Such VRF ideal functionality is implementable in the random oracle model under the CDH assumption [19].

Notation-wise, we let $\mathsf{VRF.Eval}_i^{\mathsf{ID}}(x)$ be short for $\mathsf{VRF.Eval}(sk_i, \langle \mathsf{ID}, x\rangle)$, where sk_i represents the private key of a party $\mathcal{P}_i$, and ID in our context is an explicit session identifier of a protocol instance, and also let $\mathsf{VRF.Verify}_i^{\mathsf{ID}}(x, r, \pi)$ to denote $\mathsf{VRF.Verify}(pk_i, \langle \mathsf{ID}, x\rangle, r, \pi)$.

4 Warm-up: Efficient Private-Setup Free AVSS

As briefly mentioned, asynchronous verifiable secret sharing (AVSS) is an important building block used by us for more efficient common coin and leader election. This section presents the first AVSS protocol that attains only $\mathcal{O}(\lambda n^2)$ communicated bits without relying on private setup. Somewhat surprisingly, the design is extremely simple, and it can be readily instantiated in the asynchronous message-passing model with the bulletin PKI assumption. More formally, it guarantees all the properties of the classic AVSS notion defined by Canetti and Rabin in 1993 [15] as follows:

Definition 1 (Asynchronous Verifiable Secret Sharing). *We say* (AVSS-Sh, AVSS-Rec) *realize* AVSS*, if the tuple of algorithms satisfy the following syntax and properties.*

Syntax. *For each* AVSS-Sh *instance with a session identifier* ID*, every party takes as input the system's public knowledge (i.e., λ and all public keys) and its own private keys (and there is also a designated party called dealer $\mathcal{P}_D$ that takes an additional input m called secret), and outputs a string (i.e., a secret share of $\mathcal{P}_D$'s input) at the end of the execution. For each* AVSS-Rec *instance with a session identifier* ID*, each party would input the output from the corresponding* AVSS-Sh *instance, and would collectively reconstruct the shared secret.*

Properties. (AVSS-Sh, AVSS-Rec) *shall satisfy the next properties except with negligible probability:*

- **Totality**. *If some honest party output in the* AVSS-Sh *instance associated to* ID*, then every honest party activated to execute the* AVSS-Sh *instance would complete the execution and output.*
- **Commitment**. *For any* AVSS-Sh *instance that might have any dealer, there exists a value m^* that can be fixed when some honest party output in the* AVSS-Sh *instance for* ID*. As such, when all honest parties activate* AVSS-Rec *on* ID *(with taking their corresponding* AVSS-Sh *outputs as inputs), each of them can reconstruct the same value m^*.*
- **Correctness**. *If the dealer is honest and inputs secret m in* AVSS-Sh*, then:*
  - *If all honest parties are activated to run* AVSS-Sh *on* ID*, all honest parties would output in the* AVSS-Sh *instance;*
  - *The value m^* reconstructed by any honest party in the corresponding* AVSS-Rec *instance must be equal to m, for all* ID*.*
- **Secrecy**. *In any* AVSS-Sh *instance, if the dealer is honest, the adversary shall not learn any information about the input secret from its view (which includes all internal states of corrupted parties and all messages sent to the corrupted parties). This can be formalized as that the adversary has negligible advantage in the Secrecy game (deferred to Appendix A).*

Remarks. It is worth noticing that although the above AVSS definition is quite classic in contrast to some advanced variants of AVSS (e.g., high-threshold secrecy and strong commitment), it is still powerful in the sense of empowering more efficient (reasonably fair) common randomness and consensuses in the asynchronous network environments. In particular, later Sections would demonstrate how useful this seemingly primitive AVSS definition actually can be, by showing more efficient common coin and leader election that are constructed with using this simple AVSS notion.

An efficient AVSS without private setup. The rationale behind our AVSS construction is simple. The sharing phase splits the hybrid approach of secret sharing [26] into two steps: (i) it first takes the advantage of signatures and uses linear messages to make at least $f + 1$ honest parties to agree on a polynomial commitment to an encryption key, and (ii) then the dealer multicasts a quorum proof for the completeness of the first step to convince the whole network to participate into a reliable broadcast for the ciphertext of the actual secret. While in the reconstruction phase, probably only $f + 1$ honest parties might output the polynomial commitment to the encryption key, and we cannot make them to multicast the commitment because this incurs cubic communication; nevertheless, these $f + 1$ honest

Algorithm 1 AVSS-Sh protocol with identifier ID and dealer $\mathcal{P}_D$

/* Protocol for the dealer $\mathcal{P}_D$ */

1: **upon** receiving input secret $m \in \mathbb{Z}_q$ **do**
2: choose two random polynomials $A(x)$ and $B(x)$ from $\mathbb{Z}_q[x]$ of degree at most f
3: let a_j to be the j^{th} coefficient of $A(x)$ and b_j to be that of $B(x)$ for $j \in [n]$,
4: let $key \leftarrow a_0 = A(0)$, i.e., $A(0)$ is also called key
5: compute $c_j \leftarrow g_1^{a_j} g_2^{b_j}$ for each $j \in [0, f]$, and let $C \leftarrow \{c_j\}_{j \in [0,f]}$
6: **send** KeyShare$(\mathsf{ID}, C, A(j), B(j))$ to $\mathcal{P}_j$ for each $j \in [n]$

7: **upon** receiving Stored(ID, σ_j) from $\mathcal{P}_j$ s.t. $\mathsf{SigVerify}_j^{\mathsf{ID}}(\mathcal{H}(C), \sigma_j) = 1$ **do**
8: $\Pi \leftarrow \Pi \cup \{(j, \sigma_j)\}$
9: **if** $|\Pi| = n - f$ **then**
10: $c \leftarrow key \oplus m$ and **send** Cipher$(\mathsf{ID}, \Pi, \mathcal{H}(C), c)$

/* Protocol for each party $\mathcal{P}_i$ */

11: $\mathsf{sh}_A \leftarrow \perp$, $\mathsf{sh}_B \leftarrow \perp$, $\mathsf{cmt} \leftarrow \perp$, $flag \leftarrow 0$
12: **upon** receiving KeyShare$(\mathsf{ID}, C', A'(i), B'(i))$ from $\mathcal{P}_D$ for the first time **do**
13: parse C' as $\{c'_0, c'_1, \ldots, c'_f\}$
14: **if** $g_1^{A'(i)} g_2^{B'(i)} = \prod_{k=0}^{f} {c'_k}^{i^k}$ **then**
15: record $A'(i)$, $B'(i)$ and C', $flag \leftarrow 1$, $\sigma \leftarrow \mathsf{Sign}_i^{\mathsf{ID}}(\mathcal{H}(C'))$, **send** Stored$(\mathsf{ID}, \sigma)$ to $\mathcal{P}_D$

16: **upon** receiving Cipher(ID, Π, h, c) from $\mathcal{P}_D$ for the first time **do**
17: **wait** for $flag = 1$
18: **if** $\mathcal{H}(C') = h$ and Π has $n - f$ valid signatures for h from distinct parties **then**
19: $\mathsf{sh}_A \leftarrow A'(i)$, $\mathsf{sh}_B \leftarrow B'(i)$ and $\mathsf{cmt} \leftarrow C'$
20: **send** Echo(ID, h, c) to all parties

21: **upon** receiving $2f + 1$ Echo(ID, h, c) from distinct parties **do**
22: **send** Ready(ID, h, c) to all parties if Ready not sent yet

23: **upon** receiving $f + 1$ Ready(ID, h, c) from distinct parties **do**
24: **send** Ready(ID, h, c) to all parties if Ready not sent yet

25: **upon** receiving $2f + 1$ Ready(ID, h, c) from distinct parties **do**
26: **output** $(h, c, \mathsf{sh}_A, \mathsf{sh}_B, \mathsf{cmt})$

parties can recover the encryption key, so we let them multicast the key to ensure all honest parties to finally obtain the same key and successfully decrypt the secret to output.

Here we elaborate our new AVSS protocol in details. Let us begin with the sharing protocol AVSS-Sh (cf. formal description in Alg. 1) that proceeds in the following two main logic phases:

1. *Key sharing* (Line 1-6, 11-15). In this phase, the dealer distributes the key shares to all parties using Pedersen's VSS scheme [31]. The dealer begins by randomly constructing two polynomial $A(x)$ and $B(x)$ of degree at most f. Let $A(0) = key$. Then, the dealer computes a commitment $C = \{c_j\}$ to $A(x)$, where each element $c_j = g_1^{a_j} g_2^{b_j}$, and a_j and b_j represent the j^{th} coefficients of $A(x)$ and $B(x)$ respectively. The dealer sends a KeyShare message to each party $\mathcal{P}_j$ containing the commitment C as well as $A(j)$ and $B(j)$. After receiving KeyShare message from the dealer, $\mathcal{P}_i$ checks the commitment C with $A(i)$ and $B(i)$, and records them with sending a Stored message containing a signature for $\mathcal{H}(C)$ to the dealer.
2. *Cipher broadcast* (Line 7-10, 16-26). In this phase, the dealer broadcasts its actual input encrypted by the key shared in the earlier phase, i.e., $A(0)$. After receiving $n - f$ valid Stored messages from different parties, the dealer sends Cipher to all parties containing a ciphertext c encrypting its input m, the hash digest h of the polynomial commitment C, and a quorum proof Π containing $n - f$ valid signatures for h from distinct parties. The remaining process of the phase is similar to a Bracha's reliable broadcast for h and c [10], except that each party has to "validate" h due to Π carried by the Cipher message to further proceed. At the end of the phase, each party can output $(h, c, A(i), B(i), C)$, where $A(i)$, $B(i)$ and C can be $\perp$.

Algorithm 2 $\mathsf{AVSS\text{-}Rec}$ protocol with identifier ID, for each party $\mathcal{P}_i$

Initialization: $\Phi \leftarrow \emptyset$

1: **upon** being activated with input $(h, c, \mathsf{sh}_A, \mathsf{sh}_B, \mathsf{cmt})$ **do**
2:     **if** $\mathsf{cmt} \neq \perp$, $\mathsf{sh}_A \neq \perp$ and $\mathsf{sh}_B \neq \perp$ **then**
3:         **send** $\text{KeyRec}(\mathsf{sh}_A, \mathsf{sh}_B, h)$ to all parties

4: **upon** receiving $\text{KeyRec}(\mathsf{sh}_{A,j}, \mathsf{sh}_{B,j}, h_j)$ from $\mathcal{P}_j$ for the first time **do**
5:     **if** $\mathsf{cmt} \neq \perp$ and $\mathcal{H}(\mathsf{cmt}) = h_j$ **then**
6:         parse cmt as $\{c_0, c_1, \ldots, c_f\}$
7:         **if** $g_1^{\mathsf{sh}_{A,j}} g_2^{\mathsf{sh}_{B,j}} = \prod_{k=0}^{f} c_k{}^{j^k}$ **then**
8:             $\Phi \leftarrow \Phi \cup (j, \mathsf{sh}_{A,j})$
9:             **if** $|\Phi| = f + 1$ **then**
10:                 interpolate polynomial $A(x)$ from Φ and compute $key \leftarrow A(0)$
11:                 **send** $\text{Key}(\mathsf{ID}, key)$ to all parties

12: **upon** receiving $f + 1$ Key messages containing the same key **do**
13:     $m \leftarrow key \oplus c$
14:     **output** m

Now we introduce AVSS-Rec (cf. formal description in Alg. 2) that proceeds in two logic phases:

1. *Key recovery* (Line 1-11). For each party $\mathcal{P}_i$ that activates AVSS-Rec with taking the output of AVSS-Sh as input, it sends KeyRec message containing $A(i)$, $B(i)$, and the hash h of commitment C, in case these variables are not $\perp$. Then, at least $f + 1$ honest parties, which already received the commitment C consistent to h, can solicit enough shares of the polynomial $A(x)$ committed to C through the KeyRec messages, and then interpolate the shares to obtain $A(x)$ and compute the encryption key $A(0)$.
2. *Key amplification* (Line 12-14). Upon obtaining the encryption key, each party multicast it, thus all honest parties can receives $f + 1$ Key messages containing the same key, so they can compute $m = key \oplus c$ and output m.

Note that the AVSS-Sh and AVSS-Rec protocols are parameterized by two generators g_1 and g_2 of a cyclic group $\mathbb{G}_q$ s.t. the underlying discrete logarithm problem is intractable.

Theorem 1. *The algorithms shown in Alg. 1 and Alg. 2 realize* AVSS *as defined in Definition 1, in the asynchronous message-passing model with $n/3$ static Byzantine corruption and bulletin PKI assumption without private setups, conditioned on the hardness of DLog problem and the collision-resistance of cryptographic hash function.*

We deferred the (quite standard) proof of this theorem to Appendix B, while the intuition of proving the above simple AVSS protocol is clear: the totality is mainly because our construction employs Bracha broadcast's message pattern for distributing the ciphertext encrypting the input secret and the hash of polynomial commitment; the secrecy follows the information theoretic argument due to Pedersen [31] about his verifiable secret sharing; the commitment is ensured by Pedersen commitment, collision-resistant hash, and the unforgeability of signatures.

Complexities of AVSS. The complexities of the AVSS shown in Alg. 1 and Alg. 2 can be easily seen:

– *Message.* The message complexity is $O(n^2)$ in both AVSS-Sh and AVSS-Rec protocols, since each party sends n Echo and Ready messages in AVSS-Sh and at least $f + 1$ parties send n KeyRec and Key messages in AVSS-Rec.
– *Communication.* Assuming that the input secret is of $\mathcal{O}(\lambda)$ bits. There are $O(n)$ messages with $\mathcal{O}(\lambda n)$ bits and $O(n^2)$ messages with $\mathcal{O}(\lambda)$ bits, thus the communication complexity of the protocol is of overall $O(\lambda n^2)$ bits.
– *Running-time.* The running time of AVSS is constant. Specifically, there are 5 rounds in AVSS-Sh and 2 rounds in AVSS-Rec.

Further efficiency improvement. Remark that assuming the random oracle model, one can further reduce the number of needed public key operations in Algorithm 1 and 2. The key idea is to replace "$c \leftarrow key \oplus m$" (the line 10 of Alg. 1) and "$m \leftarrow key \oplus c$" (the line 13 of Alg. 2) by "$c \leftarrow \mathcal{H}(key) \oplus m$" and "$m \leftarrow \mathcal{H}(key) \oplus c$", respectively, where $\mathcal{H}$ is a random oracle. After the replacement, the polynomial $B(x)$ is no longer needed to hide key behind the polynomial commitment C, and thus all operations related to $B(x)$ can be removed. The proof for AVSS secrecy can be similarly argued, because gaining non-negligible advantage in the AVSS's secrecy-game indicates that the adversary can break Dlog assumption to query RO with the right key in a polynomial number of queries.

Moreover, our simple AVSS protocol and the technique to construct it could be of practical merits besides its theoretic improvements. In particular, we remark that the seemingly dominating computational cost incurred by line 7 in Algorithm 2 (the reconstruction phase) actually can be pre-computed upon receiving the polynomial commitment C in Algorithm 1 (the sharing phase), indicating that each party's actual on-line computational cost could be restricted to a linear number of cheap group operations at best. This hints its real-world applications, e.g., construct concretely more efficient asynchronous casual broadcast [12,29,21] (which is a stronger variant of asynchronous atomic broadcast ensuring that the output transactions remains confidential, before they are surely finalized to output).

5 Efficient ABA without Private Setup

This section presents a novel way to private-setup free ABA. At the core of the design, it is a new reasonably fair common coin protocol that can be instantiated by AVSS along with using VRFs in the bulletin PKI setting. The Coin scheme attains constant running time, $\mathcal{O}(n^3)$ messages and $\mathcal{O}(\lambda n^3)$ bits. Thus, many existing ABAs [18,30] can adopt it for reducing private setup, and preserve constant running time and optimal resilience, while attaining at most cubic communications.

5.1 Efficient common coin without private setup

One of our aims is to design a private-setup free common coin (Coin) protocol that costs only $\mathcal{O}(n^3)$ messages and $\mathcal{O}(\lambda n^3)$ communicated bits. Formally, we would design such an efficient protocol among n parties that can tolerate up to f static corruptions, thus realizing the properties of common coin defined as follows, in the asynchronous message-passing model with the PKI assumption:

Definition 2 ($(n, f, f{+}k, \alpha)$-Common Coin). *A protocol realizes $(n, f, f{+}k, \alpha)$-Coin, if it is executed among n parties with up to f static byzantine corruptions and has syntax and properties as follows.*

Syntax. *For each protocol instance with session identifier ID, every party takes the system's public knowledge (i.e., λ and all public keys) and its own private keys as input, and outputs a single bit.*

Properties. *It satisfies the next properties except with negligible probability:*

- **Termination**. *If all honest parties are activated on ID, every honest party will output a bit for ID.*
- **Unpredictability**. *Prior to that k honest parties ($1 \le k \le f+1$) are activated to execute the protocol on ID, the adversary $\mathcal{A}$ cannot fully predicate the output. More precisely, consider the following predication game: $\mathcal{A}$ guesses a bit b^* before k honest parties activated on ID, if b^* equals to some honest party's output for ID, we require that* $\Pr[\mathcal{A}$ predicts $b^*] \le 1 - \alpha/2$.

Remarks. With at least α probability taken over all possible Coin executions, the adversary cannot predict the output coin better than guessing. A Coin protocol is said to be perfect, if $\alpha = 1$. Nevertheless, many (binary) Byzantine agreements (ABA) do not necessarily need perfect Coin to be efficient as well as optimal resilient. Many classic ABA constructions [15,30] actually can remain expected constant running time as long as using a $(n, f, f+1, \alpha)$-Coin scheme, where α is a certain constant smaller than 1. Also, sometimes it can be important to realize larger k (e.g., $n-2f$) to clip the power of asynchronous adversary in binary agreement, for example, Cachin et al. in [13] pointed out this as desiderata to save a round of communication in their ABA construction. Remark that the common coin protocol presented in this Section is $(n, f, 2f+1, 1/3)$-Coin, which is strictly stronger than $(n, f, f+1, 1/3)$-Coin.

Tackling the seed of VRF. Our Coin construction might rely on VRF to reduce the number of needed AVSS. Before elaborating our Coin construction, it is worth mentioning an issue of this cryptographic primitive in the PKI setting. Different from many studies that implicitly assume the private key of VRF is generated by a trusted third-party [17], we opt out of any private setup including such trusted key generation for VRFs. That means, a compromised party can execute a corrupted key generation procedure and register maliciously chosen VRF keys, and probably can bias the distribution of its VRF evaluations during the Coin protocol. Imagine that: if the VRF input (a.k.a. seed) used in the Coin protocol are predictable by the adversary, it becomes feasible for a corrupted party to repeat its VRF's key generation for several times to choose a private-public key pair that is more favorable (e.g., to bias the most significant bits of its VRF evaluation).

Trusted nonce from "genesis". In many settings, this might not be an issue, since there could be a trusted nonce string generated after all parties have registered their VRF keys (e.g., the same rationale behind the "genesis block" in some Proof-of-Stake blockchains [19]), which can be naturally used as the VRF seed. Such a functionality was earlier formalized by David, Gazi, Kiayias and Russell in [19] as a trusted initialization functionality to output a fully unpredictable VRF seed.

Generating nonce on the fly. Nevertheless, in some other cases, we do not have such a trusted VRF seed from "genesis", and therefore have to handle the issue by generating unpredictable VRF seeds on the fly during the course of the coin protocol. To this end, we put forth a new notion called *led reliable seeding* (Seeding) as amendment to bootstrap VRF with unpredictable nonce. Intuitively, the notion can be understood as a "broadcast" version of perfect common coin without termination in case of malicious leader. Formally, the Seeding protocol can be defined as follows.

Definition 3 (Led Reliable Seeding). Seeding *has syntax and properties defined as follows.*

SYNTAX. Seeding *is a two-stage protocol consisting of a committing phase and a revealing phase. For each protocol instance with an identifier* ID*, it is with a designated party called leader* $\mathcal{P}_L$ *and is executed among* n *parties with up to* f *static byzantine corruptions. Each party takes as input the system's public knowledge and its private keys, and then sequentially executes the committing phase and the revealing phase, at the end of which it outputs a* λ*-bit string seed.*

PROPERTIES. *It satisfies the next properties with all but negligible probability:*

– **Totality**. *If some honest party output in the* Seeding *instance associated to* ID*, then every honest party activated to execute the* Seeding *instance would complete the execution and output.*
– **Correctness**. *For all* ID*, if the leader* $\mathcal{P}_L$ *is honest and all honest parties are activated on* ID*, all honest parties would output for* ID*.*
– **Commitment**. *Upon any honest party completes the protocol's committing phase and starts to run the protocol's revealing phase on session* ID*, there exists a fixed value seed, such that if any honest party outputs for* ID*, then it outputs value seed .*
– **Unpredictability**. *Prior to that* k *honest parties (*$1 \leq k \leq f+1$*) are activated to run the protocol's revealing phase on session* ID*, the adversary* $\mathcal{A}$ *cannot predicate the output seed. Namely,* $\mathcal{A}$ *guesses a value seed** *before the honest party is activated on* ID*, then the probability that seed** *= seed shall be negligible, where seed is the output of some honest party for* ID*.*

REMARKS. Combining the commitment and unpredictability properties would ensure that no one can predicate the output, before the value to output has already been fixed, which is critical to guarantee that the evaluations of VRFs cannot be biased by exploiting the seed generation. Intuitively, if the adversary still can bias its own VRF's output on evaluating the output *seed*, it must can query the VRF oracle (which performs as random oracle [19]) with the right *seed* in a number of polynomial queries (before *seed* is committed), which would raise contradiction to break the unpredictability.

Lemma 1. *In the asynchronous message-passing model with bulletin PKI assumption, there exists a* Seeding *protocol among* n *parties that can tolerate up to* $f = n/3$ *static byzantine corruptions, terminate in constant asynchronous rounds, and cost only* $\mathcal{O}(n^2)$ *messages and* $\mathcal{O}(\lambda n^2)$ *bits, under the SXDH assumption and the collision-resistance of cryptographic hash function.*

Given the recent elegant result of aggregatable public verifiable secret sharing (PVSS) due to Gurkan et al. [22], constructing an exemplary Seeding protocol and proving its security are rather tedious, and thus we defer the proof for Lemma 1 along with the exemplary construction to Appendix C. Intuitively, it is simple to let each party send an aggregatable PVSS script carrying a random secret to the leader, so the leading party can aggregate them to produce an aggregated PVSS script committing an unpredictable nonce contributed by enough parties (e.g., $2f+1$). Then, before recovering the unpredictable secret hidden behind the aggregated PVSS script, the leader must send it to at least $2f+1$ parties to collect enough digital signatures to form a "certificate" to prove that the nonce is fixed and committed to the PVSS script. Only after seeing such the proof, each party would decrypt its corresponding share from the committed PVSS script, thus ensuring the unpredictability and commitment properties.

Algorithm 3 Coin protocol with identifier ID, for each party $\mathcal{P}_i$

$S \leftarrow \emptyset$, $\Sigma \leftarrow \emptyset$, $R \leftarrow \emptyset$, $C \leftarrow \emptyset$, $X \leftarrow 0$

$\widetilde{S} \leftarrow \perp$, $\widehat{S} \leftarrow \perp$, $seed_j \leftarrow \perp$ for each j in $[n]$

activate $\mathsf{Seeding}[\langle \mathsf{ID}, j\rangle]$ for each $j \in [n]$ with being the leader in $\mathsf{Seeding}[\langle \mathsf{ID}, i\rangle]$

  ▷ If a trusted nonce η is provided by "genesis" (generated by an initialization functionality after all parties have registered at PKI, cf. [19]), this line can be replaced by "$seed_j \leftarrow \eta$, for each $j \in [n]$"

upon $seed_j \leftarrow \mathsf{Seeding}[\langle \mathsf{ID}, j\rangle]$ **do**

  if $j = i$ **then**

    $(r, \pi) \leftarrow \mathsf{VRF.Eval}_i^{\mathsf{ID}}(seed_j)$, and activate $\mathsf{AVSS\text{-}Sh}[\langle \mathsf{ID}, j\rangle]$ as dealer with taking (r, π) as input

  else

    activate $\mathsf{AVSS\text{-}Sh}[\langle \mathsf{ID}, j\rangle]$ as a non-dealer participant

upon receiving output from $\mathsf{AVSS\text{-}Sh}[\langle \mathsf{ID}, j\rangle]$ **do**

  $S \leftarrow S \cup \{j\}$

  if $|S| = n - f$ **then**

    $\widetilde{S} \leftarrow S$ and **send** $\textsc{Lock}(\mathsf{ID}, \widetilde{S})$ to all parties

upon receiving $\textsc{Lock}(\mathsf{ID}, \widetilde{S}_j)$ from $\mathcal{P}_j$ for the first time **do**

  upon $\widetilde{S}_j \subseteq S$ **do**

    $\sigma^j \leftarrow \mathsf{Sign}_i^{\mathsf{ID}}(\mathcal{H}(\widetilde{S}_j))$ and **send** $\textsc{Confirm}(\mathsf{ID}, \sigma^j)$ to $\mathcal{P}_j$

upon receiving $\textsc{Confirm}(\mathsf{ID}, \sigma_j^i)$ from $\mathcal{P}_j$ s.t. $\mathsf{SigVerify}_j^{\mathsf{ID}}(\mathcal{H}(\widetilde{S}), \sigma_j^i) = 1$ **do**

  $\Sigma \leftarrow \Sigma \cup \{j, \sigma_j^i\}$

  upon $|\Sigma| = n - f$ **do**

    send $\textsc{Commit}(\mathsf{ID}, \Sigma, \mathcal{H}(S))$ to all parties

upon receiving $\textsc{Commit}(\mathsf{ID}, \Sigma_j, h_j)$ message from $\mathcal{P}_j$ for the first time **do**

  if $|\{k \mid (k, \sigma_k^j) \in \Sigma_j\}| = n - f \wedge \forall (k, \sigma_k^j) \in \Sigma_j$, $\mathsf{SigVerify}_k^{\mathsf{ID}}(h_j, \sigma_k^j) = 1$ **then**

    $\widehat{S} \leftarrow S$ and **send** $\textsc{RecRequest}(\mathsf{ID}, k)$ to all parties for every $k \in \widehat{S}$

    wait for that for every $k \in \widehat{S}$, $\mathsf{AVSS\text{-}Rec}[\langle \mathsf{ID}, k\rangle]$ outputs (r_k, π_k) **do**

      for each $k \in \widehat{S}$ **do**

        if $\mathsf{VRF.Verify}_k^{\mathsf{ID}}(seed_k, r_k, \pi_k) = 1$ **then**

          $R \leftarrow R \cup (k, r_k, \pi_k)$

      if $R \neq \emptyset$ **then** $\ell \leftarrow \text{argmax}_k\{r_k \mid (k, r_k, \pi_k) \in R\}$ ▷ Index of the largest VRF in R

      else $\ell \leftarrow \perp$, $r_\ell \leftarrow \perp$, $\pi_\ell \leftarrow \perp$

      send $\textsc{Candidate}(\mathsf{ID}, \ell, r_\ell, \pi_\ell)$ to all parties

upon receiving $\textsc{RecRequest}(\mathsf{ID}, k)$ from any party for the first time **do**

  wait for $\widehat{S} \neq \perp$ and that $\mathsf{AVSS\text{-}Sh}[\langle \mathsf{ID}, k\rangle]$ outputs ss_k **do** ▷ If $\widehat{S}$ becomes $\emptyset$, it is no longer $\perp$.

    activate $\mathsf{AVSS\text{-}Rec}[\langle \mathsf{ID}, k\rangle]$ with taking ss_k as input

upon receiving $\textsc{Candidate}(\mathsf{ID}, \ell', r_{\ell'}, \pi_{\ell'})$ from $\mathcal{P}_j$ for the first time **do**

  if $\ell' = \perp$ **then** $X \leftarrow X + 1$

  else if $\mathsf{VRF.Verify}_{\ell'}^{\mathsf{ID}}(seed_{\ell'}, r_{\ell'}, \pi_{\ell'}) = 1$ **then** ▷ Verifying VRF implicitly waits for $seed_{\ell'} \neq \perp$

    $C \leftarrow C \cup (j, \ell', r_{\ell'}, \pi_{\ell'})$

  if $|C| + X = n - f$ **then**

    $\hat{\ell} \leftarrow \text{argmax}_{\hat{\ell}}\{r_{\hat{\ell}} | (j, \hat{\ell}, r_{\hat{\ell}}, \pi_{\hat{\ell}}) \in C\}$ ▷ Index of the largest VRF in C

    output the lowest bit of $r_{\hat{\ell}}$

Constructing Coin without private setup. Having Seeding at hand, we are ready to present our new Coin construction (formally shown in Alg. 3), which has four main logic phases:

1. *VRFs sharing* (Line 1-8). Each party activates a Seeding process as leader and participants in all other Seeding processes to get the seeds. A party $\mathcal{P}_i$ activates its own AVSS-Sh instance as dealer to share its VRF evaluation-proof after obtaining its own VRF seed $seed_i$; once obtaining $seed_j$ besides $seed_i$, $\mathcal{P}_i$ also activates the corresponding AVSS-Sh instance as a participant.
2. *VRFs commitment* (Line 9-19). Each party $\mathcal{P}_i$ records a local set S of indices representing the completed AVSS-Sh instances. Once the local S of $\mathcal{P}_i$ contains $n-f$ indices, it takes a "snapshot" $\widetilde{S}$ of its local S and multicasts $\widetilde{S}$ to all parties via LOCK messages. Upon receiving a LOCK message from some party $\mathcal{P}_j$ containing $\widetilde{S}_j$, $\mathcal{P}_i$ waits for its local S eventually becomes a superset of $\widetilde{S}_j$, and returns a signature for $\widetilde{S}_j$ to $\mathcal{P}_j$. Thus, each party $\mathcal{P}_i$ can collect a proof Σ containing $n-f$ signatures for its "snapshot" $\widetilde{S}$ and then multicast Σ to all parties via COMMIT messages.
3. *VRFs revealing* (Line 20-32). After receiving the first COMMIT message with a valid Σ, a party $\mathcal{P}_i$ can immediately start to reconstruct AVSS associated to the indices in its current local S. After reconstructions, it might get some valid VRF evaluation-proof, and then selects the one with maximum evaluation out of the VRF evaluation-proof pairs, and multicasts this to all parties via CANDIDATE messages.
4. *Output amplification* (Line 33-39). After receiving $n-f$ CANDIDATE messages encapsulating valid VRF evaluation-proof pairs, $\mathcal{P}_i$ selects the largest evaluation and outputs the lowest bit.

Theorem 2. *In the the bulletin PKI setting, the algorithm shown in Alg. 3 realizes* $(n, f, 2f+1, 1/3)$-Coin *against* $n/3$ *static Byzantine corruptions in the asynchronous message-passing model, conditioned on that the underlying primitives are all secure.*

The proofs of this Theorem are somewhat straightforward. We defer the details to Appendix D. The security intuition is essentially straightforward: the Seeding protocols provide unpredictable nonce for VRFs, so each party computes its VRF unbiasedly. Moreover, using AVSSes to share VRF evaluations would not leak the honest parties' VRF evaluations to the adversary, so the adversary cannot prevent $f+1$ honest parties from reaching a common joint set of $n-f$ AVSSes, which has at least $\alpha = 1/3$ probability to commit the largest VRF evaluation that is also computed by some honest party. With the same probability α, all honest parties would output the lowest bit of this largest VRF, which also cannot be predicated by the adversary because all honest parties' VRF evaluations are confidential before reconstructing AVSSes.

Complexities of Coin. The complexities of the Coin protocol can be easily seen as follows:

- *Message.* The Coin protocol activates n AVSS and n Seeding instances, each of which incurs $O(n^2)$ messages. In addition, each party sends n LOCK, CONFIRM, and CANDIDATE messages. So the overall message complexity of Coin is $O(n^3)$.
- *Communication.* Each AVSS and Seeding instance in the Coin protocol exchanges $O(\lambda n^2)$ bits, since the input secret remains $O(\lambda)$ bits. The message size of LOCK and CANDIDATE is $O(\lambda n)$ and the message size of CONFIRM is $O(\lambda)$. Thus, the overall communication of Coin is $O(\lambda n^3)$.
- *Running time.* The running time of Coin is constant, which is trivial to see as the underlying AVSS-Sh and AVSS-Rec instances can terminated in constant time deterministically.

5.2 Resulting ABA without private setup

Given the new private-setup free Coin protocol, one can construct more efficient asynchronous binary agreement (ABA) with expected constant running time and cubic communications with PKI only. In particular, we primarily focus on ABA with the following standard definition [13,30].

Definition 4 (Asynchronous Binary Agreement). *A protocol realizes* ABA*, if it has syntax and properties defined as follows.*

SYNTAX. *For each protocol instance with an identifier* ID*, each party input a single bit besides some implicit input including all parties public keys and its private key, and outputs a bit* b.

PROPERTIES. *It satisfies the next properties with all but negligible probability:*

– **Termination**. *If all honest parties activate on* ID*, then every honest party outputs for* ID*.*
– **Agreement**. *Any two honest parties that output associated to* ID *would output the same bit.*
– **Validity**. *If any honest party outputs* b *for* ID*, then at least an honest party inputs* b *to* ABA[ID]*.*

Constructing ABA **without private setup**. We refrain from reintroducing the ABA protocols presented in [30] and [18], as we only need to plug in our Coin primitive to instantiate their reasonably fair common coin abstraction. More formally,

Theorem 3. *Given our* $(n, f, 2f + 1, 1/3)$-Coin *protocol, [30] and [18] implement* ABA *in the asynchronous message-passing model with* $n/3$ *static Byzantine corruption and bulletin PKI assumption, and they cost expected constant running-time, expected* $\mathcal{O}(n^3)$ *messages and expected* $\mathcal{O}(\lambda n^3)$ *bits.*

The proofs for termination, agreement and validity can be found in [30] and [18], respectively. The complexities of resulting ABA implementations would be dominated by our $(n, f, 2f + 1, 1/3)$-Coin protocol, because given costless Coin, both [30] and [18] would attain expected constant running-time, expected $\mathcal{O}(n^2)$ messages and expected $\mathcal{O}(\lambda n^2)$ bits.

6 Efficient VBA without Private Setup

This Section presents our efficient asynchronous leader election (Election) protocol without relying on private setup, which is the key step to realize fast, efficient and private-setup free multi-valued validated byzantine agreement (VBA) in the asynchronous network environment. Considering that VBA is the quintessential core building block for implementing efficient asynchronous atomic broadcast [21], our technique might initial a novel path to easy-to-deploy replicated services in the asynchronous network. In addition, the result in this section can be plugged in the AJM+21 ADKG protocol to reduce its communication complexity from $\mathcal{O}(\lambda n^3 \log n)$ to $\mathcal{O}(\lambda n^3)$.

6.1 Efficient leader election without private setup

The aim of the Election primitive is to randomly elect an index out of the indices of all participating parties [3]; more importantly, the adversary shall not be able to fully predicate which party would be elected before some honest parties start to run the protocol. Formally, we would design an Election protocol attaining the following properties in the asynchronous network:

Definition 5 **($(n, f, f + k, \beta)$-Leader Election).** *A protocol is said to be* $(n, f, f + k, \beta)$-Election*, if it is among* n *parties with up to* f *static byzantine corruptions, and has syntax and properties as follows.*

Syntax. *For each protocol instance with session identifier* ID*, every party takes the system's public knowledge (i.e.,* λ *and all public keys) and its own private keys as input, and outputs a value* $\ell \in [n]$*.*

Properties. *It satisfies the following properties except with negligible probability:*

– **Termination**. *Conditioned on that all honest parties are activated on* ID*, every honest party would output a value* $\ell \in [n]$*.*
– **Agreement**. *For any two honest parties* $\mathcal{P}_i$ *and* $\mathcal{P}_j$ *that output* ℓ_i *and* ℓ_j *for* ID*, respectively, there is* $\ell_i = \ell_j$*.*
– **Unpredictability**. *Before* k *honest parties* $(1 \leq k \leq f + 1)$ *are activated on* ID*, the adversary* $\mathcal{A}$ *cannot fully predicate the output* ℓ*. More precisely, consider the following predication game: the adversary* $\mathcal{A}$ *guesses a value* ℓ^* *before* k *honest parties activated on* ID*, if* ℓ^* *equals to some honest party's output for* ID*, we say that* $\mathcal{A}$ *wins; we let the adversary's advantage in the predication game to be* $|\Pr[\mathcal{A} \text{ wins}] - \beta/n\ |$ *and require it at most* $1 - \beta$*.*

Remarks. With at least β probability taken over all possible Election executions, the adversary cannot predict the output index better than guessing over $[1, n]$. Different from Coin that does not have perfect agreement, Election requires any two honest parties to output the same index. This is important in many validated Byzantine agreement (VBA) constructions, because the Election's output is usually used to directly decide that the final output is from which party's proposal, so lacking agreement in

Election might immediately break the agreement in VBA. When plugging in a $(n, f, f+k, \beta)$-Election implementation with agreement, most VBA constructions [12,3,28] can preserve their securities, as long as β is a certain constant between $(0, 1]$. Also, sometimes it can be important to realize larger k (e.g., $f+1$) to clip the attacking power of asynchronous adversary as if in the VBA construction of [28]. The leader election protocol presented in this Section is a $(n, f, 2f+1, 1/3)$-Election protocol, which is strictly stronger than $(n, f, f+1, 1/3)$-Election.

Constructing Election without private setup. Our Election protocol is formally shown in Alg. 4, and it has three main logic phases that can be explained as follows:

1. *Committing largest VRF* (Line 1-4). In this phase, each party firstly executes the code of Coin protocol, and obtains the speculative largest VRF evaluation seen in its view, i.e., get $\mathsf{rnd}_{\mathsf{max}} = (\ell^*, r^*, \pi^*)$ where ℓ^* represents that this largest VRF is computed by which party, and r^* and π^* are the VRF evaluation and proof, respectively. After that, the party broadcasts $\mathsf{rnd}_{\mathsf{max}}$ to commit the largest VRF evaluation in its view. Here Bracha's reliable broadcasts (RBC) [10] are used to prevent from committing different speculative largest VRF to distinct parties.
2. *Voting on how to output* (Line 5-13). After a party receives $n-f$ RBC outputs that include a valid VRF evaluation-proof, it checks if there exists a RBC output that: (i) it carries a VRF evaluation same to the majority of $n-f$ RBC outputs', and (ii) it also carries the largest VRF evaluation among all $n-f$ VRF evaluations received from RBCs. If such an element exists, the party takes a "snapshot" of these $n-f$ RBCs outputs to form G^*, then it sends $\textsc{Vote}(\mathsf{ID}, G^*)$ to all parties and activates ABA with input 1, otherwise, it activates ABA with input 0.
3. *Output decision* (Line 14-17). If ABA outputs 1, each party waits for any $\textsc{Vote}(\mathsf{ID}, G^*)$ message[3] that carries a G^* such that (i) G^* is a subset of its received RBC outputs and (ii) there exists r^* that is the largest and majority VRF evaluation among all elements in G^*, then it outputs $(r^* \bmod n) + 1$. If ABA outputs 0, all parties would output a default index, e.g., 1.

Algorithm 4 Election protocol with identifier ID, for each party $\mathcal{P}_i$

1: $G \leftarrow \emptyset$, $G^* \leftarrow \emptyset$, $\mathit{ballot} \leftarrow 0$, activate $\mathsf{RBC}[\langle \mathsf{ID}, j\rangle]$ for each $j \in [n]$
2: run the code of Coin in Alg. 3 with replacing Line 39 by "$\mathsf{rnd}_{\mathsf{max}} \leftarrow (\hat{\ell}, r_{\hat{\ell}}, \pi_{\hat{\ell}})$"
3: **wait** for $\mathsf{rnd}_{\mathsf{max}}$ assigned by (ℓ^*, r^*, π^*) // i.e., Line 39 of modified Alg. 3 is executed
4:     input (ℓ^*, r^*, π^*) to $\mathsf{RBC}[\langle \mathsf{ID}, i\rangle]$

5: **upon** $\mathsf{RBC}[\langle \mathsf{ID}, j\rangle]$ outputs (ℓ^*, r^*, π^*) **do**
6:     **if** $\mathsf{VRF.Verify}^{\mathsf{ID}}_{\ell^*}(\mathit{seed}_{\ell^*}, r^*, \pi^*) = 1$ **then** ▷ Verifying VRF implicitly waits for $\mathit{seed}_{\ell^*} \neq \perp$
7:         $G \leftarrow G \cup (j, \ell^*, r^*, \pi^*)$
8:         **if** $|G| = n - f$ **then**
9:             **if** exist $(\cdot, \ell^*, r^*, \cdot)$ matching the majority elements in G **then**
10:                 **if** r^* is the largest VRF evaluation among all elements in G **then**
▷ Namely, there exists r^* that is the largest and majority VRF among G
11:                     $G^* \leftarrow G$, $\mathit{ballot} \leftarrow 1$, and **send** $\textsc{Vote}(\mathsf{ID}, G^*)$ to all parties
12:             activate $\mathsf{ABA}[\mathsf{ID}]$ with ballot as input
13:             **wait** for that $\mathsf{ABA}[\mathsf{ID}]$ outputs b
14:                 **if** $b = 1$ **then**
15:                     **wait** for any $\textsc{Vote}(\mathsf{ID}, G^*)$ message where $G^* \subset G$ and there exists the largest and majority VRF evaluation r^* among the $n-f$ elements in G^* ▷ Verifying VRF implicitly waits for *seed*
16:                         **output** $(r^* \bmod n) + 1$, where r^* is the largest and majority VRF among G^*
17:                 **else output** the default index, i.e., 1

---

[3] Remark that we use the $\textsc{Vote}(\mathsf{ID}, G^*)$ message for presentation brevity. Actually, a party does not have to send and wait $\textsc{Vote}$ message, because we can specify some quite involved "rules" used by the honest parties to check their local set G to decide output upon ABA returns 1.

Theorem 4. *In the bulletin PKI setting, Alg. 4 realizes* $(n, f, 2f+1, 1/3)$-Election *in the asynchronous message-passing model against* $n/3$ *static Byzantine corruptions, conditioned on that the underlying primitives are all secure.*

We defer the tedious details of proving this Theorem to Appendix E. Given our Coin construction, the main barrier of lifting it to Election is to ensure perfect agreement while preserving the other benefits of our Coin protocol. Intuitively, we go through the challenge by combining ABA and our specific voting rules: everyone broadcasts the speculative largest VRF evaluation obtained from Coin and waits for receiving $n-f$ such broadcasted VRFs to check (i) if there exists a majority one in them and (ii) if the majority one is also the largest one. Noticing that ABA would output 1, only if an honest party realizes the above voting rules are satisfied, and there must be a unique VRF evaluation satisfying the voting rules. In addition, with at least $1/3$ probability, the Coin execution would be lucky to make all honest parties realize a common largest VRF evaluated by an honest party; in such case, ABA always returns 1 as no honest party inputs 0 to ABA, and then the low bits of the VRF evaluation becomes the elected leader.

Complexities of Election. The complexities of the Election protocol can be easily seen as follows:

- *Message.* The execution of Coin part spends $O(n^3)$ messages, n RBC incurs $O(n^3)$ messages, the ABA instance costs expected $O(n^3)$ messages, and there are $O(n^2)$ VOTE messages in total. Thus, the overall message complexity is expected $O(n^3)$.
- *Communication.* The Coin part costs $O(\lambda n^3)$ bits as argued earlier, each RBC instance inputs $O(\lambda)$ bits and thus costs $O(\lambda n^2)$ bits, the ABA instance incurs expected $O(\lambda n^3)$ bits, and the size of each VOTE message is $O(\lambda n)$ bits. So Election exchanges $O(\lambda n^3)$ bits on average.
- *Running-time.* The Election protocol can terminate in expected $O(1)$ asynchronous rounds.

6.2 Resulting VBA without private setup

Given our Election protocol, we are ready to construct the private-setup free validated Byzantine agreement (VBA), which is an important building block for implementing asynchronous atomic broadcast [21,12,27,33] and is essentially a special Byzantine agreement with external validity that can be defined as follows.

Definition 6 (Asynchronous Validated Byzantine Agreement). *A protocol realizes* VBA*, if it has syntax and properties defined as follows.*

SYNTAX. *For each protocol instance with an identifier* ID *and a global polynomial-time computable predicate* Q_{ID}*, each party input a value besides some implicit input including all parties public keys and its private key, and outputs one value.*

PROPERTIES. *It satisfies the next properties with all but negligible probability:*

- **Termination**. *If all honest parties activate on* ID *with an input satisfying* Q_{ID}*, then every honest party outputs for* ID.
- **Agreement**. *Any two honest parties that output associated to* ID *would output the same value.*
- **External-Validity**. *If any honest party outputs* v *for* ID*, then* $Q_{\mathsf{ID}}(v) = 1$.

Constructing VBA without private setup. Most existing VBA constructions [12,3,28] rely on a trusted dealer to faithfully distribute the private key shares of a non-interactive threshold PRF (tPRF) [13] to implement a Leader Election primitive that can uniformly elect a common index out of all parties. Fortunately, our reasonably fair Election protocol is pluggable in all VBA implementations [12,3,28] to replace such tPRFs, thus removing the unpleasant private setup. More formally,

Theorem 5. *In the bulletin PKI setting, given our* $(n, f, 2f+1, 1/3)$-Election *protocol, [12,3,28] would implement* VBA *in the asynchronous message-passing model with* $n/3$ *static Byzantine corruption, and they would cost expected constant running-time, expected* $\mathcal{O}(n^3)$ *messages and expected* $\mathcal{O}(\lambda n^3)$ *bits for* λn*-bit input.*

Proofs for VBA properties can be found in [12,3,28] with some trivial adaptions. Here we briefly discuss the intuition behind the securities and the resulting asymptotic complexities with using Abraham et al.'s VBA construction [3] as an example.

Recall that each iteration of Abraham et al.'s VBA [3] proceeds as follows: every party begins as a leader to perform a 4-stage provable-broadcast (PB) protocol to broadcast a **key** proof (carrying a value as well), a **lock** proof, and a **commit** proof, where each proof is essentially a quorum certificate; following **key-lock-commit** proofs, each party can further generate and multicast a completeness proof, attesting that it delivers these proofs to at least $f+1$ honest parties, which is called leader nomination; then, after at least $n-f$ 4-staged PBs are proven to complete leader nominations, a Election primitive is needed to sample a party called leader in a perfect fair (or reasonably fair) way; so with some constant probability, i.e., $2/9$ in case of plugging our $(n, f, 2f+1, 1/3)$-Election protocol, the elected leader already finished its nomination and delivered a **commit** proof to at least $f+1$ honest parties, and these parties can output the value received from the leader's 4-stage PB; and after one more round to multicast and amplify the proofs, all parties also output the same value; otherwise, it is a worse case with $7/9$ probability, in which no enough honest parties **commit** regarding the elected leader, and the protocol enters the next iteration; nonetheless, the nice properties of the **key** and **lock** proofs would ensure that the parties can luckily output in the next iteration with the same $2/9$ chance.

Thus, plugging our Election primitive into Abraham et al.'s VBA would preserve its constant running time. For message complexity, no extra cost is placed except our Election primitive, so it becomes dominated by the $\mathcal{O}(n^3)$ messages of Election. For communication complexity, it is worth noticing that non-interactive threshold signature scheme is used to form short quorum certificates in the 4-staged provable-broadcast (PB) protocols; nevertheless, such instantiation of quorum certificate can be replaced by trivially concatenating digital signatures from $n-f$ distinct parties in the bulletin PKI setting, which only adds an $\mathcal{O}(n)$ factor to the size of quorum certificates, thus causing $\mathcal{O}(\lambda n^3)$ communication complexity to this private-setup free VBA instantiation (for λn-bit input).

Application to asynchronous distributed key generation. Given our new path to constructing efficient VBA protocols with $\mathcal{O}(\lambda n^3)$ communication complexity and constant running time, one can simply use it to replace the VBA instantiation in AJM+21 [2] to improve the same authors' asynchronous distributed key generation, which would reduce the communication cost from $\mathcal{O}(\lambda n^3 \log n)$ to $\mathcal{O}(\lambda n^3)$ with preserving all other benefits such as constant running time and optimal resilience.

References

1. Abraham, I., Aguilera, M.K., Malkhi, D.: Fast asynchronous consensus with optimal resilience. In: International Symposium on Distributed Computing. pp. 4–19 (2010)
2. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Reaching consensus for asynchronous distributed key generation. arXiv preprint arXiv:2102.09041 (2021)
3. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 337–346 (2019)
4. AlHaddad, N., Varia, M., Zhang, H.: High-threshold avss with optimal communication complexity. In: International Conference on Financial Cryptography and Data Security (2021)
5. Attiya, H., Welch, J.: Distributed computing: fundamentals, simulations, and advanced topics, vol. 19. John Wiley & Sons (2004)
6. Backes, M., Datta, A., Kate, A.: Asynchronous computational vss with reduced communication complexity. In: Topics in Cryptology – CT-RSA 2013. pp. 259–276
7. Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: Advances in Cryptology – ASIACRYPT 2011. pp. 590–609
8. Bangalore, L., Choudhury, A., Patra, A.: Almost-surely terminating asynchronous byzantine agreement revisited. In: Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing. pp. 295–304 (2018)
9. Blum, E., Katz, J., Liu-Zhang, C.D., Loss, J.: Asynchronous byzantine agreement with subquadratic communication. In: Theory of Cryptography Conference. pp. 353–380 (2020)
10. Bracha, G.: Asynchronous byzantine agreement protocols. Information and Computation **75**(2), 130–143 (1987)

11. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. pp. 88–97 (2002)
12. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)
13. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: practical asynchronous byzantine agreement using cryptography. In: 19th Annual ACM Symposium on Principles of Distributed Computing (2000)
14. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05). pp. 191–201. IEEE (2005)
15. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing. pp. 42–51 (1993)
16. Cascudo, I., David, B.: SCRAPE: Scalable randomness attested by public entities. In: Proc. ACNS 2017. pp. 537–556
17. Cohen, S., Keidar, I., Spiegelman, A.: Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. In: 34th International Symposium on Distributed Computing (DISC 2020) (2020)
18. Crain, T.: Two more algorithms for randomized signature-free asynchronous binary byzantine consensus with $t < n/3$ and $\mathcal{O}(n^2)$messages and $\mathcal{O}(1)$ round expected termination. arXiv preprint arXiv:2002.08765 (2020)
19. 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 (2018)
20. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Tech. rep., Massachusetts Inst of Tech Cambridge lab for Computer Science (1982)
21. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 803–818 (2020)
22. Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Aggregatable distributed key generation. In: Advances in Cryptology – EUROCRYPT 2021
23. Kate, A., Miller, A., Yurek, T.: Brief note: Asynchronous verifiable secret sharing with optimal resilience and linear amortized overhead. arXiv preprint arXiv:1902.06095 (2019)
24. Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. arXiv preprint arXiv:2102.08325 (2021)
25. Kokoris Kogias, E., Malkhi, D., Spiegelman, A.: Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1751–1767 (2020)
26. Krawczyk, H.: Secret sharing made short. In: Advances in Cryptology – CRYPTO 1993. pp. 136–146
27. Kursawe, K., Shoup, V.: Optimistic asynchronous atomic broadcast. In: International Colloquium on Automata, Languages, and Programming. pp. 204–215. Springer (2005)
28. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: Proceedings of the 39th Symposium on Principles of Distributed Computing. pp. 129–138 (2020)
29. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 31–42. ACM (2016)
30. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous binary byzantine consensus with $t < n/3$, $\mathcal{O}(n^2)$ messages, and $\mathcal{O}(1)$ expected time. Journal of the ACM (JACM) **62**(4), 31 (2015)
31. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140 (1991)
32. Rabin, M.O.: Randomized byzantine generals. In: 24th Annual Symposium on Foundations of Computer Science (sfcs 1983). pp. 403–409. IEEE (1983)
33. Ramasamy, H.V., Cachin, C.: Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In: International Conference On Principles Of Distributed Systems. pp. 88–102. Springer (2005)
34. Reiter, M.K.: Secure agreement protocols: Reliable and atomic group multicast in rampart. In: Proceedings of the 2nd ACM Conference on Computer and Communications Security. pp. 68–80 (1994)

Appendix A Secrecy Game for AVSS

Secrecy Game. *The Secrecy game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is defined as follows to capture the secrecy threat in the* AVSS *protocol among n parties with up to f static corruptions in the bulletin PKI setting:*

1. *$\mathcal{A}$ chooses a set $\bar{Q}$ of up to f parties to corrupt, a dealer $\mathcal{P}_D$ ($\mathcal{P}_D \notin \bar{Q}$) and a session identifier* ID*, and also generates the secret-public key pairs for each corrupted party in $\bar{Q}$, and sends $\bar{Q}$, $\mathcal{P}_D$,* ID *and all relevant public keys to $\mathcal{C}$.*
2. *The challenger $\mathcal{C}$ generates the secret-public key pair for every honest party in $[n] \setminus \bar{Q}$, and sends these public keys to the adversary $\mathcal{A}$.*
3. *$\mathcal{A}$ chooses two secrets s_0 and s_1 with same length and send them to $\mathcal{C}$.*
4. *The challenger $\mathcal{C}$ decides a hidden bit $b \in \{0, 1\}$ randomly, executes the* AVSS-Sh *protocol on* ID *(for all honest parties) to share s_b via interacting with $\mathcal{A}$ (that are on behalf of the corrupted parties). During the execution, $\mathcal{A}$ is consulted to schedule all message deliveries and would learn: (i) the protocol scripts sent to the "corrupted parties", and (ii) the length of all messages sent among the honest parties.*
5. *The adversary $\mathcal{A}$ guesses a bit b'.*

The advantage of $\mathcal{A}$ in the above Secrecy game $\mathbf{Adv_{sec}}$ *is* $|\Pr[b = b'] - 1/2|$.

Recall that the secrecy requirement of AVSS requires that the adversary's advantage in the above game shall be negligible.

Appendix B Deferred Proof for AVSS Construction

Lemma 2. *If any two honest parties $\mathcal{P}_i$ and $\mathcal{P}_j$ output $(h, c, \cdot, \cdot, \cdot)$ and $(h', c', \cdot, \cdot, \cdot)$ in* AVSS-Sh[ID], *respectively, then $h = h'$ and $c = c'$ except with negligible probability.*

Proof. Suppose that $h \neq h'$ (resp. $c \neq c'$), $\mathcal{P}_i$ receives $2f + 1$ Ready message containing h (resp. c), the senders of which include at least $f + 1$ honest parties; in the same way, $\mathcal{P}_j$ must have received at least $f + 1$ Ready messages containing h'(resp. c') from honest parties; so it induces that at least one honest party sent two different messages, which is impossible. So there is a contradiction if $h \neq h'$ or $c \neq c'$, implying $h = h'$ and $c = c'$.

Deferred proof for Theorem 1. Here down below we prove Theorem 1, namely, to analyze the AVSS protocol presented by Alg. 1 and Alg. 2 in details to argue how it securely realizes Def. 1.

Proof. Here prove that Alg. 1 and Alg. 2 satisfy AVSS's properties one by one:

- *Totality.* Assume that an honest party outputs in the AVSS-Sh, it must have received $2f+1$ Ready messages. At least $f + 1$ of the messages are sent from honest parties. Therefore, all parties will eventually receive $f + 1$ Ready messages from these $f + 1$ honest parties and send a Ready message as well. Then, all honest parties will eventually receive $2f + 1$ valid Ready messages and output. So the totality property always holds.
- *Commitment.* From Lemma 2, all honest parties that complete AVSS-Sh would agree on the same h and c. According to the collision-resistance of hash function, the adversary cannot find a $C' \neq C$ such that $h = \mathcal{H}(C') = \mathcal{H}(C)$ with all but negligible probability, so there is a fixed C except with negligible probability. Moreover, C is computationally binding conditioned on DLog assumption, so all honest parties agree on the same polynomial $A^*(x)$ committed to C, which fixes a unique key^*, and they also receive the same cipher c. So there exists a unique $m^* = c \oplus key^*$, which can be fixed once some honest party outputs in AVSS-Sh.
  Now we prove that m^* can be reconstructed when all honest parties activate AVSS-Rec. Any honest party outputs in the AVSS-Sh subprotocol must receive $2f + 1$ Ready messages from distinct parties, at least $f+1$ of which are from honest parties. Thus, at least one honest party has received $2f+1$ Echo messages from distinct parties. This ensures that at least $f+1$ honest parties get the same commitment C and a valid quorum proof $\mathit{\Pi}$. Due the unforgeability of signatures in $\mathit{\Pi}$, that means at least $f+1$ honest parties did store valid shares of $A^*(x)$ and $B^*(x)$ along with the corresponding commitment C except with negligible probability. So after all honest parties start AVSS-Rec, there are at least $f + 1$ honest parties would broadcast KeyRec messages with valid shares of $A^*(x)$ and $B^*(x)$. These messages can be received by all parties and can be verified by at least $f + 1$ honest parties who record C. With overwhelming probability, at least $f + 1$ parties can

interpolate $A^*(x)$ to compute $A^*(0)$ as key and broadcast it, and all parties can receive at least $f+1$ same key^* and then output the same $m^* = c \oplus key^*$ as they obtain the same ciphertext c from AVSS-Sh.

– *Correctness.* For proving the first part, it is clear that (i) the honest dealer must collect at least $n-f$ valid digital signature for $\mathcal{H}(C)$ from distinct parties to form valid Π and (ii) every honest party can eventually wait the shares of $A(x)$ and $B(x)$ as well as the same C. This implies that all honest parties can eventually broadcast the same Cipher messages, so they would broadcast the same Echo messages and the same Ready messages, thus finally outputting in the AVSS-Sh instance.

  For proving the second party, it is easy to see that (i) any honest party must output a ciphertext c same to the ciphertext computed by the honest sender and (ii) all honest parties must receive the same hash h of the commitment C to $A(x)$, where $A(x)$ is a polynomial chosen by the honest deader. Recall that we have proven that all honest parties can reconstruct a message $c \oplus A(0)$, which exactly is m because c computed by the honest sender is $m \oplus A(0)$.

– *Secrecy.* The adversary's $view$ in an AVSS-Sh execution with an honest dealer would include the commitment C (and its hash h), the ciphertext c, some signatures for h, the secret shares received by up to f corrupted parties as well as all public keys and corrupted parties secret keys. The signatures leak nothing related to the input secret (even if the adversary can fully break digital signature to learn the private signing keys). Thus, following the information-theoretic argument in [31], the adversary's advantage in the Secrecy game $\mathbf{Adv_{sec}} = |\Pr[b = b' \mid view] - 1/2\,| = |\Pr[b' = 0]\Pr[b = 0] + \Pr[b' = 1]\Pr[b = 1] - 1/2\,| = 1/2 - 1/2 = 0$ because the commitment C is perfectly hiding and f shares of Shamir's secret sharing scheme also leaks nothing about the input secret.

Appendix C An Implementation of Seed Generation

Here we give an exemplary construction for reliable leaded seeding (Seeding) through the elegant idea of aggregatable public verifiable secret sharing (PVSS) in [22]. The general idea of [22] is to lift the beautiful Scrape PVSS scheme [16] to enable the aggregation of PVSS scripts from distinct participating parties, and along the way, it presents a way of using knowledge-of-signatures to allow each party to attach an aggregatable "tag" attesting its contribution in the aggregated PVSS script, thus ensuring that anyone can check the finally aggregated PVSS script pvss (and tag) to verify whether pvss indeed commits a polynomial collectively "generated" by more than f parties while preserving the size of communicated scripts minimal.

It thus becomes immediate to follow the nice idea to construct an efficient Seeding protocol as shown in Alg. 5 in the PKI setting, in which: (i) each party firstly generates a Scrape PVSS script along with a knowledge-of-signature, such that the leader can collect and aggregate $2f+1$ Scrape PVSS scripts and multicast the aggregated PVSS script along with a vector of knowledge-of-signatures and some other metadata (called tag by us) to the whole network, later (ii) each party returns to the leader a signature for the aggregated PVSS script, so the leader can collect and multicast a quorum certificate containing at least $2f+1$ signatures to attest that it has "committed" a consistent PVSS script collectively contributed by at least $2f+1$ parties across the whole network, finally (iii) it becomes simple for every party to multicast its own secret share regarding to the final PVSS script, thus reconstructing the secret collectively generated by at least $2f+1$ parties, which is naturally the random seed to output.

For sake of completeness, we briefly review the cryptographic abstraction of the aggregatable PVSS scheme (with unforgeable tags attesting contributions) among n parties with a secrecy threshold t due to Gurkan et al. [22]. Note that we mainly focus on abstracting the needed properties to construct and prove our Seeding protocol.

C.1 PVSS

Let us begin with introducing the syntax and properties PVSS scheme without aggregability. Remark that the scheme might need some public parameter param (which is simply the description for some cyclic groups and would be explained later). Under the PKI setting, each party $\mathcal{P}_i$ is bounded to an encryption-decryption key pair (ek_i, dk_i). The decryption key dk_i is only known by the party $\mathcal{P}_i$, while the encryption key ek_i is known by all parties. Let ek denote the encryption keys $\{ek_1, \ldots, ek_n\}$ of all parties.

Syntax. The (n, t)-PVSS scheme can be described as a tuple of non-interactive algorithms as follows (with taking param as an implicit input):

- $\mathsf{Deal}(\mathsf{ek}, s) \to \mathsf{pvss}$. This (probably probabilistic) algorithm takes a secret s as input and outputs the PVSS script pvss.
- $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{pvss}) \to 0/1$. This deterministic algorithm takes all encryption keys as input, thus that it can verify whether a PVSS script pvss is valid in the sense that pvss commits a fixed polynomial that can be reconstructed collectively by n parties (i.e., output 1) or not (i.e., 0).
- $\mathsf{GetShare}(dk_i, \mathsf{pvss}) \to \mathsf{sh}_i$. When executed by $\mathcal{P}_i$, this algorithm takes a valid pvss script and the party's decryption key dk_i as input and outputs the secret share sh_i regarding the secret committed to pvss.
- $\mathsf{VrfyShare}(j, \mathsf{sh}_j, \mathsf{pvss}) \to 0/1$. It takes a PVSS script pvss and a secret share sh_j from $\mathcal{P}_j$, and it verifies whether sh_j is the correct j^{th} share of the polynomial committed to pvss (i.e., output 1) or not (i.e., output 0).
- $\mathsf{AggShares}(\{(j, \mathsf{sh}_j)\}_t) \to a$. It takes t valid secret shares from distinct parties regarding an implicit PVSS script pvss and computes the secret a committed to the implicit pvss.
- $\mathsf{VrfySecret}(s, \mathsf{pvss}) \to 0/1$. It verifies whether a secret s is indeed committed to pvss (i.e., output 1) or not (i.e., output 0).

Properties. It satisfies *correctness*, *public verifiability* and *secrecy*:

- **Correctness** is as straightforward as follows:
  - $\forall\ s \in \mathbb{Z}_q$, the script verification passes for an honest dealer, namely,

$$\Pr\left[\ \mathsf{VrfyScript}(\mathsf{ek}, \mathsf{pvss}) = 1 \mid \mathsf{pvss} \leftarrow \mathsf{Deal}(\mathsf{ek}, s)\ \right] = 1$$

  - Any honestly decrypted share of any valid PVSS script can be validated,

$$\Pr\left[\mathsf{VrfyShare}(j, \mathsf{sh}_j, \mathsf{pvss}) = 1 \;\middle|\; \begin{array}{l} \mathsf{VrfyScript}(\mathsf{ek}, \mathsf{pvss}) = 1 \\ \mathsf{sh}_j \leftarrow \mathsf{GetShare}(dk_j, \mathsf{pvss}) \end{array}\right] = 1$$

  - If an honest dealer shares a secret s with a PVSS script pvss, then any t honest parties (e.g., denoted by a set Q) can collectively reconstruct the original secret s, namely,

$$\Pr\left[\mathsf{AggShares}(S) = s \;\middle|\; \begin{array}{l} S \leftarrow \{(j, \mathsf{sh}_j \leftarrow \mathsf{GetShare}(dk_j, \mathsf{pvss}))\}_{j \in Q} \\ Q \in \{P \mid P \in [n] \wedge |P| = t\} \\ \mathsf{pvss} \leftarrow \mathsf{Deal}(\mathsf{ek}, s) \end{array}\right] = 1$$

  - The secret obtained through aggregating t valid secret shares for a valid pvss script can also be validated with using the same pvss, namely,

$$\Pr\left[\mathsf{VrfySecret}(s, \mathsf{pvss}) = 1 \;\middle|\; \begin{array}{l} Q \in \{P \mid P \in [n] \wedge |P| = t\} \\ \mathsf{VrfyShare}(j, \mathsf{sh}_j, \mathsf{pvss}) = 1, \forall j \in Q \\ \mathsf{AggShares}(S) = s \wedge S \leftarrow \{(j, \mathsf{sh}_j)\}_{j \in Q} \end{array}\right] = 1$$

- **Publicly verifiable commitment** indicates that any sharing script pvss can be verified by the public to tell whether it is valid to commit a fixed secret $F^*(0)$ or not. Namely, if pvss is valid due to $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{pvss}) = 1$, it is guaranteed that:
  - There exists a fixed secret $F^*(0)$, such that $\mathsf{VrfySecret}(F^*(0), \mathsf{pvss}) = 1$.
  - Any t decryption shares validated by $\mathsf{VrfyShare}$ with regard to the pvss script from distinct parties (including up to f malicious ones) can be combined to yield a secret $F(0)$ same to $F^*(0)$.
- **Weak secrecy** means that in our setting, the probability that the adversary wins the Weak Secrecy game down below is negligible. Note that the weak secrecy well captures the scenario that: the honest dealer uniformly chooses a secret over $\mathbb{Z}_q$ to share, and it is computationally infeasible for the adversary to compute the dealer's input secret on receiving the PVSS script carrying the secret, unless $t - f$ honest parties send their decrypted secret shares the adversary.

Weak-Secrecy game. *The Weak-Secrecy game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is defined as follows to capture the secrecy threat in (n,t)-*PVSS *scheme with up to f static corruptions ($f+1 \leq t$):*

1. *The adversary $\mathcal{A}$ generates the decryption-encryption key pairs for all up to f corrupted parties, and sends these public keys to the challenger $\mathcal{C}$.*
2. *The challenger $\mathcal{C}$ generates the decryption-encryption key pairs for all honest parties, and sends these public keys to the adversary $\mathcal{A}$.*
3. *The challenger $\mathcal{C}$ chooses a secret s randomly from the space of secret $\mathbb{Z}_q$. $\mathcal{C}$ executes* $\mathsf{Deal}(\mathsf{ek}, s)$ *(where* ek *are partially generated by $\mathcal{C}$ and partially chosen by $\mathcal{A}$ in the first two steps) to get the sharing script* pvss*, and sends $\mathcal{A}$ the script* pvss*.*
4. *The adversary $\mathcal{A}$ queries $\mathcal{C}$ to get $t-f-1$ decrypted shares of* pvss*, and computes s'.*

The adversary $\mathcal{A}$ wins the above Weak-Secrecy game if $s = s'$.

C.2 Aggregatable PVSS with weight tags

To lift PVSS scheme to enjoy aggregability, an aggregating algorithm can be added to the scheme [22], which naturally combines any two valid sharing scripts to form another one:

– $\mathsf{AggScripts}(\mathsf{pvss}_1, \mathsf{pvss}_2) \rightarrow \mathsf{pvss}$. This algorithm takes two valid PVSS scripts pvss_1 and pvss_2 as input and outputs a valid PVSS script pvss. We might let $\mathsf{AggScripts}(\{\mathsf{pvss}_i, \ldots, \mathsf{pvss}_j\})$ to be short for iteratively aggregating the pvss scripts in the set.

Now we expect the *commitment* property can be extended accordingly to capture the aggregation of PVSS scripts, that is:

– Suppose that two valid PVSS scripts pvss_1 and pvss_2 commit two secrets $F_1^*(0)$ and $F_2^*(0)$, respectively; then $\mathsf{AggScripts}(\mathsf{pvss}_1, \mathsf{pvss}_2)$ always outputs a valid pvss committing a secret $F^*(0) = F_1^*(0) + F_2^*(0)$.

The aggregatable PVSS scheme can be further lifted with weight tags (for the original purpose of gossip-based distributed key generation) in [22], which means each aggregated PVSS script pvss is attached with some implicit tags, so they can be checked by the public to verify the script pvss indeed aggregates which parties' PVSS scripts. Those lifted algorithms were termed as aggregatable distributed key generation in [22]; nevertheless, we remain to view it as PVSS with advanced properties for presentation simplicity (which is also pointed out in [22]), since we do not use it as DKG for any threshold public-key cryptosystem and just use the scheme for sharing some secret confidentially in a publicly verifiable manner. For the purpose, one can add another algorithm Weights to the PVSS scheme and also slightly lift the syntax of PVSS scheme as follows:

– $\mathsf{Deal}(\mathsf{ek}, sk_i, s) \rightarrow \mathsf{pvss}$. Now the algorithm takes an extra secret key sk_i as input when it is executed by the party $\mathcal{P}_i$, which is needed to make pvss carry an unforgeable weight tag bounded to the identity $\mathcal{P}_i$. Remark that sometimes, we might let Deal to share a randomly chosen element in $\mathbb{Z}_q$ and thus omit the input field of s to rewrite it as $\mathsf{Deal}(\mathsf{ek}, sk_i) \rightarrow \mathsf{pvss}$ for short.
– $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{vk}, \mathsf{pvss}) \rightarrow 0/1$. It takes some verification keys vk besides ek and pvss as input. The output still represents whether pvss is valid or not.
– $\mathsf{Weights}(\mathsf{pvss}) \rightarrow \vec{w}$. It takes a valid pvss script as input and outputs an n-sized vector $\vec{w}$, every j^{th} element in which belongs to $\mathbb{N}^0$ and represents that the pvss script indeed aggregates a certain pvss script from the party $\mathcal{P}_j$.

The further augmented aggregatable PVSS scheme with weight tags $\vec{w}$ shall satisfy extra properties to make $\vec{w}$ reflect the indices of the parties actually contributing in pvss. The additional *correctness* property related to weights are:

– If $\mathsf{Deal}(\mathsf{ek}, sk_i, s) \rightarrow \mathsf{pvss}$ and $\mathsf{Weights}(\mathsf{pvss}) \rightarrow \vec{w}$, then the i^{th} element at $\vec{w}$ is 1, and all else positions at $\vec{w}$ are 0, which means the script pvss is contributed by $\mathcal{P}_i$.
– If pvss_1 and pvss_2 are two valid PVSS scripts and $\mathsf{AggScripts}(\mathsf{pvss}_1, \mathsf{pvss}_2)$ outputs pvss, then $\vec{w} = \vec{w}_1 + \vec{w}_2$, where $\vec{w} \leftarrow \mathsf{Weights}(\mathsf{pvss})$, $\vec{w}_1 \leftarrow \mathsf{Weights}(\mathsf{pvss}_1)$, and $\vec{w}_2 \leftarrow \mathsf{Weights}(\mathsf{pvss}_2)$.

Unpredictability. Finally, the weights carried by each pvss script shall also be *unforgeable*, in the sense of attesting that the $\mathsf{Weights}$ algorithm can extract from the script to tell that the script indeed aggregates which parties' pvss scripts. More formally, we can think of the property by requiring the adversary to have negligible probably to win the following Unpredictability game, which intuitively lifts the aforementioned weak-secrecy to take the weight tags into consideration and states that if each honest party $\mathcal{P}_i$ randomly chooses the input secret s_j committed to its PVSS script pvss_j, then the adversary cannot compute the aggregated secret s committed to a valid pvss, as long as the script pvss has a non-zero weight to reflect some honest party's contribution in it.

Unpredictability game. *The Unpredictability game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is defined as follows for a (n, t) aggregatable* PVSS *scheme with weight tags in the presence of up to f static corruptions ($f + 1 \leq t$):*

1. *The adversary chooses a set $\bar{Q}$ of f corrupted parties, generates the public-private key pairs for each corrupted party in $\bar{Q}$, and sends all the public keys to the challenger $\mathcal{C}$.*
2. *The challenger $\mathcal{C}$ generates all public-private key pairs for all honest parties, and sends these public keys to the adversary $\mathcal{A}$.*
3. *The adversary $\mathcal{A}$ queries $\mathcal{C}$ for each party $\mathcal{P}_i$ in $[n] \setminus \bar{Q}$, such that $\mathcal{C}$ randomly chooses $s_i \in \mathbb{Z}_q$, computes $\mathsf{pvss}_i \leftarrow \mathsf{Deal}(\mathsf{ek}, sk_i, s_i)$ and sends pvss_i to $\mathcal{A}$. The adversary now produces a valid* pvss *script such that* $\mathsf{Weights}(\mathsf{pvss})$ *outputs $\vec{w}$ containing more than $f+k$ positions (where $1 \leq k \leq n-f$), and then sends* pvss *to $\mathcal{C}$.*
4. *The adversary asks the challenger to compute at most $t - f - 1$ decryption shares on the received* pvss *and then send these shares back, and then the adversary guesses a value $s^* \in \mathbb{Z}_q$.*

The adversary wins if $s^* = s$, where s is the actual secret committed to pvss *(which is unique due to the commitment property).*

Remarkably, Gurkan et al. [22] give a beautiful construction of aggregatable PVSS scheme in the random oracle model with unforgeable tags to attest the contributing parties, which satisfy all aforementioned properties needed by us for constructing a $\mathsf{Seeding}$ protocol.[4] The scheme assumes some public parameter param consisting of: (i) two cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ where SXDH assumption holds; (ii) a random generator $g \in \mathbb{G}_1$ and two random elements $h \in \mathbb{G}_2$ and $u \in \mathbb{G}_2$. Besides those, their scheme can be established in the PKI setting, as it just requires each party to register two public keys. Moreover, each PVSS script or tag includes at most $\mathcal{O}(\lambda n)$ bits. We refer the interested reader to [22] and [16] for the detailed construction.

C.3 Constructing Seeding

Here down below we describe an exemplary $\mathsf{Seeding}$ construction from the $(n, 2f + 1)$ aggregatable PVSS scheme (as detailed by Alg. 5). Recall that the protocol is a two-phase protocol (including a committing phase and a revealing phase), and its execution can be briefly described as follows:

- *Committing phase-Seed aggregation* (Line 1-4, 20-24).In this phase, each party invokes Deal to create a pvss script and send this script to the leader $\mathcal{P}_L$. When the leader receives pvss_j from $\mathcal{P}_j$, it verifies the pvss using ek and checks if the weights of pvss_j are all zeros but one at the j^{th} position. Once collecting $2f + 1$ valid pvss scripts, the leader aggregates them and send it using a LockAggPvss message.
- *Committing phase-Seed commitment* (Line 5-10, 25-29). After receiving a LockAggPvss message with a valid pvss. Each party sign for it and send the signature σ_i to the leader to commit the same pvss. Upon receiving $2f+1$ valid signature for the pvss, the leader send a CommitAggPvss message

[4] Remark that Gurkan et al. did not formalize their aggregatable PVSS scheme in our way as an aggregatable PVSS with unforgeable weight tags, since their final goal is to realize a gossip-based distributed key generation protocol, so they focus on key properties to emulate a "trusted" key generation functionality. Nevertheless, their construction can be easily argued to satisfy the properties abstracted by us for constructing $\mathsf{Seeding}$, because failing to satisfy our requirements such as Unpredictability would directly break their DKG (since the adversary might directly tell the final secret key).

containing $\mathcal{H}(\mathsf{pvss})$ and a signature set Σ. After receiving a valid CommitAggPvss messages from the leader, each party confirms that the output is actually fixed, and it takes the pvss as a input in $\mathsf{GetShare}$ to output a share sh_i regarding the secret committed to pvss.

– *Revealing phase-Seed recovery* (Line 11-19, 30-35). The leader collects $2f+1$ valid shares which are committed to pvss and aggregates them to a *seed*. Then the *seed* is sent to all parties with $\mathcal{H}(\mathsf{pvss})$ and Σ. Once honest parties receiving a valid seed with a hash value of pvss and $2f+1$ valid signatures for pvss, the execution is just like a reliable broadcast. Specifically, each party sends SeedEcho to all parties. After receiving $2f+1$ SeedEcho messages, honest parties send SeedReady to all parties. If an honest party receives $f+1$ SeedReady messages of same value and SeedReady has not been sent, it sends SeedReady. When receiving $2f+1$ SeedReady messages of same value, honest parties output the *seed*.

Algorithm 5 $\mathsf{Seeding}$ protocol with identifier ID and leader $\mathcal{P}_L$

/* Protocol for each party $\mathcal{P}_i$ */

1: **upon** being activated **do**
2: randomly sample a secret s
3: $\mathsf{pvss}_i \leftarrow \mathsf{Deal}(\mathsf{ek}, sk_i, s)$
4: **send** PvssScript$(\mathsf{ID}, \mathsf{pvss}_i)$ to $\mathcal{P}_L$

5: **upon** receiving LockAggPvss$(\mathsf{ID}, \mathsf{pvss})$ message from $\mathcal{P}_L$ for the first time **do**
6: **if** $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{vk}, \mathsf{pvss}) = 1 \wedge$ ($\mathsf{Weights}(\mathsf{pvss})$ has $2f+1$ non-zero elements) **then**
7: $\sigma_i \leftarrow \mathsf{Sign}_i^{\mathsf{ID}}(\mathcal{H}(\mathsf{pvss}))$ and **send** ConfirmAggPvss(ID, σ_i) to $\mathcal{P}_L$

8: **upon** receiving CommitAggPvss(ID, h, Σ) from $\mathcal{P}_L$ for the first time **do**
9: **if** $\mathcal{H}(\mathsf{pvss}) = h \wedge$ (Σ contains valid signatures for h from $2f+1$ distinct parties) **then**
10: $\mathsf{sh}_i \leftarrow \mathsf{GetShare}(\mathsf{pvss})$ and **send** SeedShare$(\mathsf{ID}, \mathsf{sh}_i)$ to $\mathcal{P}_L$

11: **upon** receiving Seed$(\mathsf{ID}, h, \Sigma, seed)$ from $\mathcal{P}_L$ for the first time **do**
12: **if** $\mathcal{H}(\mathsf{pvss}) = h \wedge \mathsf{VrfySecret}(seed, \mathsf{pvss}) = 1 \wedge$ (Σ contains valid signatures for h from $2f+1$ distinct parties) **then**
13: **send** SeedEcho$(\mathsf{ID}, seed)$ to all parties

14: **upon** receiving $2f+1$ SeedEcho$(\mathsf{ID}, seed)$ with same *seed* from distinct parties **do**
15: **send** SeedReady$(\mathsf{ID}, seed)$ to all parties if SeedReady not sent yet

16: **upon** receiving $f+1$ SeedReady$(\mathsf{ID}, seed)$ with same *seed* from distinct parties **do**
17: **send** SeedReady$(\mathsf{ID}, seed)$ to all parties if SeedReady not sent yet

18: **upon** receiving $2f+1$ SeedReady$(\mathsf{ID}, seed)$ with same *seed* from distinct parties **do**
19: **output** *seed*

/* Protocol for the leader $\mathcal{P}_L$ */

20: **upon** receiving PvssScript$(\mathsf{ID}, \mathsf{pvss}_j)$ from $\mathcal{P}_j$ for the first time **do**
21: **if** $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{pvss}_j) = 1 \wedge$ (the weights of pvss_j are all zeros but one at the j^{th} position) **then**
22: $K \leftarrow K \cup \{\mathsf{pvss}_j\}$
23: **if** $|K| = 2f+1$ **then**
24: $\mathsf{pvss} \leftarrow \mathsf{AggScripts}(K)$ and **send** LockAggPvss$(\mathsf{ID}, \mathsf{pvss})$ to all parties

25: **upon** receiving ConfirmAggPvss(ID, σ_j) message from $\mathcal{P}_j$ for the first time **do**
26: **if** $\mathsf{SigVerify}_j^{\mathsf{ID}}(\mathcal{H}(\mathsf{pvss}), \sigma_j) = 1$ **then**
27: $\Sigma \leftarrow \Sigma \cup \{(j, \sigma_j)\}$
28: **if** $|\Sigma| = 2f+1$ **then**
29: **send** CommitAggPvss$(\mathsf{ID}, \mathcal{H}(\mathsf{pvss}), \Sigma)$ to all parties

30: **upon** receiving SeedShare$(\mathsf{ID}, \mathsf{sh}_j)$ message from $\mathcal{P}_j$ for the first time **do**
31: **if** $\mathsf{VrfyShare}(\mathsf{sh}_j, \mathsf{pvss}) = 1$ **then**
32: $S \leftarrow S \cup \{(j, \mathsf{sh}_j)\}$
33: **if** $|S| = 2f+1$ **then**
34: $seed \leftarrow \mathsf{AggShares}(S)$
35: **send** Seed$(\mathsf{ID}, \mathcal{H}(\mathsf{pvss}), \Sigma, seed)$ to all parties

Deferred proofs for Lemma 1. Here we prove that the protocol in Alg. 5 satisfies all properties of Reliable Leaded Seeding given in Definition 3 and analyze how does it incur only quadratic messages and communications.

Proof. Here prove that Alg. 5 satisfy the $\mathsf{Seeding}$ properties one by one:

- *Totality.* Assume that an honest party outputs in the $\mathsf{Seeding}$, it must have received $2f+1$ SeedReady messages. At least $f+1$ of the messages are sent from honest parties. Therefore, all parties will eventually receive $f+1$ SeedReady messages from these honest parties and then send a SeedReady messages as well (if a SeedReady message has not been sent yet). So the totality property always holds.
- *Correctness.* In the seed aggregation phase, the leader will collect $2f+1$ valid pvss_i scripts and aggregate them into a pvss whose weight has $2f+1$ positions to be 1. In the seed commitment phase, all honest parties sign for this pvss so that the leader can collect at least $n-f$ valid signatures for $\mathcal{H}(\mathsf{pvss})$ to form valid Σ. In the seed recovery phase, the leader can collect at least $n-f$ valid shares which are committed to the pvss they have signed for. All honest parties can receive the same $\mathcal{H}(\mathsf{pvss})$, Σ and *seed* that pass verifications. So they would broadcast the same SeedEcho message and the same SeedReady messages, thus finally outputting in the $\mathsf{Seeding}$ instance.
- *Commitment.* Assume that $\mathcal{P}_i$ is the first party who starts to run the protocol's revealing phase, it implies that $\mathcal{P}_i$ received a valid CommitAggPvss(ID, h, Σ) message from leader $\mathcal{P}_L$. If another honest $\mathcal{P}_j$ received a valid CommitAggPvss$(\mathsf{ID}, h', \Sigma')$ message from leader $\mathcal{P}_L$, where $h' \neq h$, since a valid Σ contains $2f+1$ valid signatures for a same hash value from distinct parties, it induces that at least one honest party signed for both h and h', which is impossible. Hence, when some honest party $\mathcal{P}_i$ starts to run the protocol's revealing phase, the h from any valid CommitAggPvss(ID, h, Σ) message is unique. Following the commitment of the PVSS scheme, there exists a fixed value *seed* corresponding to the pvss, where $h = \mathcal{H}(\mathsf{pvss})$.
  Suppose that some honest party outputs $seed'$ from the $\mathsf{Seeding}$. By the code, it receives $2f+1$ SeedReady messages containing $seed'$. Then at least one honest party received $2f+1$ valid SeedEcho messages with the same $seed'$ from distinct parties, which means that at least $f+1$ honest parties received valid Seed$(\mathsf{ID}, h, \Sigma, seed')$ message from the leader. From the previous analysis, no honest party will accept a $seed' \neq seed$ from $\mathcal{P}_L$ or multicast it. Thus, $seed' = seed$.
- *Unpredictability.* Prior to $f+1$ honest parties are activated to run the revealing phase of the $\mathsf{Seeding}$ protocol, the adversary can only collect at most $2f$ decryption shares for the committed pvss script. Trivially according to the Unpredictability of PVSS with weight tags, since the aggregated pvss has a weight with $2f+1$ non-zero positions, it is infeasible for the adversary to compute a $seed^* = seed$ at the moment, where *seed* is the actual secret committed to the aggregated pvss script.

The complexities can be easily seen as follows: The message complexity of $\mathsf{Seeding}$ is $O(n^2)$, which is due to each party sends n SeedEcho and SeedReady messages; considering that the input secret s and pvss both are $\mathcal{O}(\lambda)$ bits, and there are $O(n)$ messages with $\mathcal{O}(\lambda n)$ bits and $O(n^2)$ messages with $\mathcal{O}(\lambda)$ bits, thus the communication complexity of the protocol is of overall $O(\lambda n^2)$ bits.

Appendix D Deferred Proofs for Common Coin

Lemma 3. *With overwhelming probability, once the first honest party receives a valid Σ_j for an S^*, at least $f+1$ honest parties have already recorded local S including S^*. We call such S^* the core set.*

Proof. If an honest party receives a valid Σ_j with $n-f$ signatures for $\mathcal{H}(S_j)$, at least $f+1$ signatures are signed by honest parties. Note that an honest party $\mathcal{P}_i$ will sign for some S_j only if $S_j \subseteq S_i$. Thus, trivially from the unforgeability of digital signatures, with all but negligible probability, if an honest parties receives a valid Σ_j for S_j, there exists a core set $S^* = S_j$ which is subset of at least $f+1$ honest parties' local S.

Lemma 4. *Let $\mathsf{Event}_{\mathsf{good}}$ to denote a case in which a core set S^* solicits an honest party's VRF evaluation that is also largest among all parties' VRF, and the remaining case denoted by $\mathsf{Event}_{\mathsf{bad}}$ to cover all other possible executions, then the probability of the $\mathsf{Event}_{\mathsf{good}}$ occurs is $\alpha \geq 1/3$, under the ideal functionality of VRF [19] (which is realizable in the random oracle model with CDH assumption).*

Proof. From Lemma 3, at the moment when the first honest party invokes any AVSS-Rec instance, it already receives a Σ_j, and there exists a core set S^* including $2n/3$ indices, each of which represents a shared VRF's evaluation and at least $n/3$ indices out of which are shared by honest parties. Recall that the honest dealers' AVSS-Rec leaks nothing about their VRF's evaluations, so at the moment when S^* is fixed, the adversary learns nothing about honest parties' VRF evaluations. Thus, the probability that the $\mathsf{Event}_{\mathsf{good}}$ occurs is $\alpha \geq 1/3$. Otherwise, the adversary directly breaks the unpredictability of VRF by either biasing the distribution of corrupted parties' VRF evaluations (which is infeasible because according to the commitment and unpredictability of Seeding, VRF seeds generated by Seeding protocols are unpredictable before it is committed, and once the Seeding completing the committing phase, the VRF seeds are fixed) or can predicate the high bits of honest parties VRF's evaluations without accessing their secret keys.

Deferred proofs for Theorem 2.

Proof. We prove that Alg. 3 realizes the properties of Coin in Def. 2 one by one:

- *Termination.* According to the correctness and commitment of Seeding, if all honest parties participate in $\mathsf{Seeding}[\langle \mathsf{ID}, j\rangle]$, every party will get the same $seed_j$ regarding an honest $\mathcal{P}_j$. Due to the totality and commitment of Seeding, if any honest party gets $seed_j$ from $\mathsf{Seeding}[\langle \mathsf{ID}, j\rangle]$, all honest parties would obtain the same $seed_j$, despite malicious $\mathcal{P}_j$.
  Then, every honest party will eventually complete at least $n-f$ AVSS-Sh instances in which it participates and get a $(n-f)$-sized set S, because there are at least $n-f$ honest parties would complete their Seeding, compute their VRFs, and activate their AVSS-Sh instances that would finally joined by all honest parties.
  Moreover, an honest $\mathcal{P}_i$ can collect a set Σ_i containing at least $n-f$ signatures for its S_i to form Σ, every honest party will have local S covering S_i from the totality of AVSS.
  If an honest party $\mathcal{P}_i$ receives a valid Σ_j for the first time, it will fix a $\hat{S}$ and send RecRequest messages, and all honest parties would complete all AVSS-Sh instances corresponds to its $\hat{S}$ according to the totality of AVSS and then start AVSS-Rec, so all secrets corresponds to its $\hat{S}$ can be reconstructed. For any index in $\hat{S}$, there is at least one honest party indeed activates the corresponding AVSS-Sh instance, indicating this honest party must output in the corresponding Seeding. This means that for each $k \in \hat{S}$, $\mathcal{P}_i$ can output in $\mathsf{Seeding}[\langle \mathsf{ID}, k\rangle]$ to get a common $seed_k$. So for each $k \in \widehat{S}$, it can check whether (k, r_k, π_k) is validated VRF result, and picks up the maximum r_l among all valid r_k. Finally, every honest party sends the picked (l, r_l, π_l) using a Candidate message to all parties. All honest parties can eventually receive at least $n-f$ valid Candidate from different parties, because all honest parties get common VRF seeds and mutually consider each others' Candidate messages valid, and then output the lowest bit of the maximum of all r_l among valid Candidate messages.
- *Unpredictability.* From Lemma 4, the $\mathsf{Event}_{\mathsf{good}}$ occurs with a probability of 1/3. Before $f+1$ honest parties are activated to run the protocol, the adversary cannot predicate the protocol execution will fall into which case, because no one can predicate the VRF seeds and thus even the corrupted parties cannot compute their VRF evaluations. Following the same argument, from the view of any adversary before $f+1$ honest parties run the protocol, and when $\mathsf{Event}_{\mathsf{good}}$ occurs, the adversary cannot predicate the output better than guessing (i.e., succeed in predicating with 1/2 probability). Note that when $\mathsf{Event}_{\mathsf{good}}$ occurs, the lowest bit b of the largest VRF evaluation would be output of all honest parties. Therefore, the adversary's advantage in the predication game is $|\Pr[\mathcal{A} \text{ wins}] - \alpha/2\,| \leq 1-\alpha$, where $\alpha = 1/3$.

Appendix E Deferred Proofs for Leader Election

Lemma 5. *If two parties $\mathcal{P}_i$ and $\mathcal{P}_j$ sends valid* $\textsc{Vote}(\mathsf{ID}, G)$ *and valid* $\textsc{Vote}(\mathsf{ID}, G')$ *to all parties, respectively, i.e., there exists $(\cdot, \ell, r, \cdot)$ matching the majority elements in G and r is the largest VRF evaluation among all elements in G, and there exists $(\cdot, \ell', r', \cdot)$ matching the majority elements in G' and r' is the largest VRF evaluation among all elements in G', then the $(\ell, r) = (\ell', r')$.*

Proof. We prove this by contradiction. Suppose $r \neq r'$. By the code, $(\cdot, \ell, r, \cdot)$ and $(\cdot, \ell', r', \cdot)$ match the majority of G and G', respectively, which means that the number of their appearance in G and G' are at least $f+1$, respectively. Without loss of generality, we assume that $r > r'$. Note that there are $n-f$ elements in G', so at least one valid $(\cdot, \ell, r, \cdot)$ must be included in G', because all elements in G and G' are obtained via reliable broadcast that ensures agreement. Since r' is the largest VRF evaluation among all elements in G', which also means $r' > r$, which is a contradiction to the assumption. Hence $(\ell, r) = (\ell', r')$.

Deferred proof for Theorem 4.

Proof. Here we prove that Alg. 4 satisfies the properties of Election in Def. 5 one by one:

- *Termination.* From the termination of Coin, each party will output a $\mathsf{rnd}_{\mathsf{max}} = (\ell^*, r^*, \pi^*)$, then each party will broadcast its (ℓ^*, r^*, π^*) using RBC. According to the validity of RBC, each honest party can eventually collect a set G containing at least $n-f$ RBC outputs. For an honest party, if there exists $(\cdot, \ell^*, r^*, \cdot)$ matching the majority elements in G and r^* is the largest VRF evaluation among all elements in G, activates the ABA[ID] with 1 as input, otherwise, inputs 0 into ABA[ID]. According to the termination and agreement of ABA, if all honest parties participate in the ABA, then all of them will output the same bit b. If $b=0$, all honest parties output the default index, i.e., 1. If $b=1$, from the validity of ABA, at least one honest party inputs with 1, by the code, it also implies this honest party sends a valid VOTE message carrying a G^* to all. Due to that each element of G^* is the output of RBC, following the totality of RBC, each party can receive all elements in G^*. Hence, all honest parties can eventually wait for a valid VOTE message containing a G^* which $G^* \subset G$ and then output.
- *Agreement.* According to the termination and agreement of ABA, if all honest parties participate in the ABA, all of them would output from ABA with the same bit b. We analyze it in two cases: (i), If $b=0$, it is obvious that all honest parties will output the default index, i.e., 1. (ii), If $b=1$, from the validity of ABA, there is at least one honest party activates ABA with input 1, which implies that at least one valid VOTE message containing a G^* has been sent and all honest parties will receive it. According to the Lemma 5, any valid G^* sent by a VOTE message have the same $(\cdot, \ell, r, \cdot)$ which matches the majority elements in G^* and r is the largest VRF evaluation. Hence, all honest parties output the same value.
- *Unpredictability.* We use the $\mathsf{Event}_{\mathsf{good}}$ and $\mathsf{Event}_{\mathsf{bad}}$ defined in the Coin to discuss this property by two cases. Case (i): With probability β, the $\mathsf{Event}_{\mathsf{good}}$ occurs, all honest parties will output the same $\mathsf{rnd}_{\mathsf{max}} = (\ell^*, r^*, \pi^*)$. In this case, all honest parties have the same (ℓ^*, r^*, π^*) and send it by RBC. Following the validity of RBC, each honest party can receive at least $n-f$ messages from distinct RBC instances, $n-2f \geq f+1$ of which are sent by distinct honest parties and contain the same (ℓ^*, r^*, π^*). So all honest parties can collect a G, in which $(\cdot, \ell^*, r^*, \cdot)$ matches the majority elements and r^* is the largest VRF. Then all honest parties send a VOTE message and activate ABA with 1 as input. From the validity of ABA, all honest parties will output 1 from ABA. By the code, all honest parties can wait for a valid VOTE message and output $(r^* \bmod n) + 1$ from Election. Since the $\beta = \Pr[\mathsf{Event}_{\mathsf{good}}] = 1/3$ and the VRF evaluation is uniformly sampled and does not leak the adversary, until a proof can be produced to attest that a core set is fixed (which needs at least $f+1$ honest parties to participate), so each $\ell \in [n]$ has the same opportunity to be some honest party's output from Election. The probability that the adversary $\mathcal{A}$ succeeds in predicting the output ℓ which coincides with some honest party's output is no more than $\frac{\beta}{n}$. Case (ii): if the $\mathsf{Event}_{\mathsf{bad}}$ occurs, the adversary $\mathcal{A}$ might lead up to different honest parties to obtain different $\mathsf{rnd}_{\mathsf{max}}$, so that some honest parties would not be able to find the majority elements $(\cdot, \ell^*, r^*, \cdot)$, where r^* is the largest VRF evaluation in G. Nevertheless, it cannot be worse than that ABA always outputs 0, so the adversary always predicate the output in the case. In sum, the advantage of adversary in the predication game is $|\Pr[\mathcal{A} \text{ wins}] - \beta/n\ | \leq 1 - \beta$, where $\beta = 1/3$.