Efficient Asynchronous Byzantine Agreement without Private Setups

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

[1] Institute of Software, Chinese Academy of Sciences
{yingzi2019,luyuan,xujing,zhenfeng}@iscas.ac.cn
[2] School of Computer Science, The University of Sydney
zhlu9620@uni.sydney.edu.au,qiang.tang@sydney.edu.au
[3] University of Chinese Academy of Sciences

Abstract. Though recent breakthroughs greatly improved the efficiency of asynchronous Byzantine agreement (BA) protocols, they mainly focused on the setting with private setups, e.g., assuming established non-interactive threshold cryptosystems. Challenges remain to reduce the large communication complexities in the absence of such setups. For example, Abraham et al. (PODC'21) recently gave the first private-setup free construction for asynchronous validated BA (VBA) with expected $\mathcal{O}(n^3)$ messages and $\mathcal{O}(1)$ rounds, 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 parties, and λ means the cryptographic security parameter capturing the length of hash, digital signature, etc. We reduce the communication of private-setup free asynchronous BA to expected $\mathcal{O}(\lambda n^3)$ bits. At the core of our design, we present a systematic treatment of reasonably fair common randomness protocols in the asynchronous network, and proceed as:

- We give an efficient reasonably fair common coin protocol in the asynchronous setting with only PKI setup. It costs only $\mathcal{O}(\lambda n^3)$ bit and $\mathcal{O}(1)$ asynchronous rounds, and ensures that with at least 1/3 probability, all honest parties can output a common bit that is as if randomly flipped. This directly renders more efficient private-setup free asynchronous binary agreement (ABA) with expected $\mathcal{O}(\lambda n^3)$ bits and expected constant rounds.
- Then, we lift our common coin to attain perfect agreement by using a single ABA. This gives us a reasonably fair random leader election protocol with expected $\mathcal{O}(\lambda n^3)$ communication and expected constant rounds. It is pluggable in all existing VBA protocols (e.g., Cachin et al., CRYPTO'01; Abraham et al., PODC'19; Lu et al., PODC'20) to remove the needed private setup or distributed key generation (DKG). As such, the communication of private-setup free VBA is reduced to expected $\mathcal{O}(\lambda n^3)$ bits while preserving fast termination in expected $\mathcal{O}(1)$ rounds. Moreover, our result paves a generic path to private-setup free asynchronous BA protocols, as it is not restricted to merely improve Abraham et al.'s specific VBA protocol in PODC'21.

Our results and techniques could be found useful and interesting for a broad array of applications such as asynchronous DKG and DKG-free asynchronous random beacon.

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 [5,45,54,39,52,61]. Nevertheless, asynchronous protocols have to rely on randomized executions to circumvent the seminal FLP "impossibility" result [33]. In particular, to quickly decide the output in expected constant rounds of interactions, many asynchronous protocols [20,55,54,17,16,5,52,39,45,22,13,1,4] essentially need *common randomness*, given which for "costless", one at least can construct optimally resilient asynchronous Byzantine agreement (BA) protocols that cost expected $\mathcal{O}(n^2)$ messages and expected $\mathcal{O}(1)$ rounds [5,52,55,17,16,23].

However, efficient ways to implement asynchronous common randomness in practice mostly rely on different varieties of private setups. For example, initiated by M. Rabin [60], it assumes that a trusted dealer directly uses secret sharing to distribute a large number of random secrets among the participating parties before the protocol starts, so the parties can collectively reconstruct a sequence of common randomness while running the protocol. Later, Cachin et al. [17] presented how to set up a non-interactive threshold pseudorandom function (tPRF) by assuming that a trusted dealer can faithfully share a short tPRF key, which now is widely used by existing practical asynchronous BFT protocols including [5,45,54,52].

These private setups might cause unpleasant deployment hurdles, preventing asynchronous protocols from being widely used in broader settings. 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 BA, but most are costly or even prohibitively expensive.

Back to 1993, Canetti and Rabin [20] 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 invoke to let the honest parties collectively recover the earlier 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 [55,20,23], it incurs tremendous $\mathcal{O}(n^6)$ messages and $\mathcal{O}(\lambda n^8 \log n)$ bits, where λ is the cryptographic security parameter. The huge complexities of CR93 are dominated by its expensive AVSS. Since then, many more efficient private-setup free AVSS protocols [15,9,6,10] were proposed and can directly improve it. For example, Cachin et al. [15] 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) still incur $\mathcal{O}(n^3)$ messages and $\mathcal{O}(\lambda n^4)$ bits, which remains expensive and exists an $\mathcal{O}(n)$ gap between the message and the communication complexities.

Recently, Kokoris-Kogias et al. [47] (KMS20) presented a new path to reducing the common coin primitive to AVSS.[1] In particular, KMS20 incurs $\mathcal{O}(\lambda n^4)$ bits and $\mathcal{O}(n)$ asynchronous rounds to generate a single random coin, which is seemingly worse than CKLS02. Nonetheless, once being bootstrapped, it can continually generate coins at a lower per-coin cost of $\mathcal{O}(\lambda n^2)$ bits and $\mathcal{O}(1)$ rounds, thus being cheaper in an amortized way. In another recent breakthrough, Abraham et al. [4] presented an elegant asynchronous validated Byzantine agreement (VBA) protocol (AJM+21) without private setup. It costs expected $\mathcal{O}(n^3)$ messages, constant rounds, and $\mathcal{O}(|m|n^2 + \lambda n^3 \log n)$ bits for $|m|$-bit input,[2] and only assumes the presence of a bulletin PKI that can facilitate the management of public keys. At the core of AJM+21 VBA, it lifts reasonably fair common coin to a new *random proposal election* primitive, such that with a constant probability, the honest parties can randomly decide a common value proposed by some non-corrupted party. As such, a certain VBA protocol called *No-Waitin' HotStuff* (NWH) was tailored to cater for this special proposal election primitive. However, this election notion is too specific to be used in other existing VBA constructions [5,52,16], due to the imperfect of necessary agreement. Still, AJM+21 VBA costs $\mathcal{O}(\lambda n^3 \log n)$ bits, and leaves room for further reducing the communication cost asymptotically by removing the $\log n$ factor.

Bearing the state-of-the-art, it calls to systematically treat the fundamental basis of more efficient (reasonably fair) common randomness protocols, such that we can make them private-setup free and pluggable in the existing asynchronous Byzantine agreement designs, thus overcoming the current deployment hurdles of asynchronous protocols. That said, the following question remains open:

Can we design efficient asynchronous common randomness protocols with fewer setup assumptions, thus reducing the expected communication cost of asynchronous Byzantine agreements (e.g., ABA *and* VBA*) to $\mathcal{O}(\lambda n^3)$ bits?*

[1] KMS20 requires a strengthened AVSS with high-threshold secrecy: the adversary cannot learn the secret before $n - 2f$ honest parties start to reconstruct (where f is the number of corrupted parties). In contrast, the classic AVSS notion [20] only preserves secrecy, before the first honest party activates reconstruction.

[2] Through the paper, we consider the input size $|m|$ of VBA to be at most λn bits, so the $|m|n^2$ term does not dominate the communication complexity, thus ignored. For larger input, it can be an orthogonal problem to push the $|m|n^2$ term to $|m|n$, as discussed by many "extension" protocols [52,58,57,36] for multi-valued BA.

1.1 Our contribution

We give an affirmative answer to the above question. At the core of our solution, we develop a set of new techniques to design an efficient private-setup free construction for reasonably fair common coin that are pluggable in many existing ABA protocols [20,55,23]; more interestingly, we formalize and construct an efficient (reasonably fair) leader election notion with perfect agreement, by lifting our common coin protocol to be always agreed. This leader election primitive can be directly plugged in all existing VBA protocols [16,5,52,38,37] to remove their reliance on private setups.

Table 1: **Comparison of private-setup free asynchronous BA protocols**

	ABA/Coin		VBA/Election		Adaptive	Cryptographic	Setup
	Comm.	Round	Comm.	Round	Security?	Assumption	Assumption
CKLS02 [15] §	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(1)$	-	-	Yes	Dlog+hash	global param *
KMS20 [47] †	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(n)$	No⋆	RO+DDH⋆	PKI ♯
AJM+21 [4] †‡	$\mathcal{O}(\lambda n^3 \log n)$	$\mathcal{O}(1)$	$\mathcal{O}(\lambda n^3 \log n)$ ¶	$\mathcal{O}(1)$	No	RO+SXDH	PKI
This paper	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$	No	RO+SXDH	PKI
					Yes	RO+DDH	PKI, 1-time rnd **

∗ Global parameters capture some minimal setups such as an agreed group description and group generators. For some schemes relying on collision-resistant hash [15,47], a one-time common random string is needed to key the hash functions.
§ CKLS02 [15] did not construct VBA or leader election. We also do not realize any complexity-preserving reductions to it.
⋆ KMS20 states that it might be adaptively secure by using the pairing-based adaptively secure threshold signature [50], and this might cause it rely on SXDH assumption instead of only DDH assumption.
♯ The PKI setup in KMS20 can be removed by recent high-threshold AVSS presented in [6].
† Note that KMS20 and AJM+21 did not present an explicit construction for random leader election (Election). Nevertheless, they gave asynchronous distributed key generation protocols (ADKG) that can bootstrap threshold verifiable random function and thus can set up Election (and also common coin) schemes via ADKG.
‡ AJM+21 only presents an explicit VBA construction but does not construct ABA. However, VBA implies ABA with same complexities, because there is a simple complexity-preserving reduction from ABA to VBA in the PKI setting, cf. [16].
¶ The communication of AJM+21 can be reduced to $\mathcal{O}(\lambda n^3)$ by a recent reliable broadcast protocol [25], but this only applies to the specific AJM+21 VBA construction, while our result is generic and can be adapted to all existing VBA protocols.
∗∗ 1-time rnd means a one-time common random string can be published after PKI registration but before protocol execution.

In greater details, our technical contribution is three-fold:

- We implement $\mathcal{O}(\lambda n^3)$-bit and $\mathcal{O}(1)$-round **common coin and** ABA **with only PKI setup** in the asynchronous network, conditioned on SXDH assumption and random oracle.
  The crux of our design is a novel efficient construction for the reasonably fair common coin in the bulletin PKI setting (in the random oracle model). Different from CR93 (that used n^2 AVSS instances), we use verifiable random function in combination with more efficient AVSS construction to reduce the number of necessary AVSS instances by an $\mathcal{O}(n)$ order. This private-setup free common coin costs only $\mathcal{O}(\lambda n^3)$ bits and constant asynchronous rounds. With our common coin protocol at hand, we can 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 [15], while preserving other benefits such as the maximal $n/3$ resilience and the optimal expected constant rounds. Even comparing with a recent work due to Abraham et al. [4] that presents a more efficient VBA construction and improves ABA as a by-product,[3] our approach still realizes a $\log n$ factor improvement.
- We also realize $\mathcal{O}(\lambda n^3)$-bit and $\mathcal{O}(1)$-round **random leader election and** VBA **with only PKI** in the asynchronous setting, assuming SXDH assumption and random oracle.
  At the core of this contribution, we use one single ABA protocol to lift our common coin and clean up the possible disagreement among honest parties, and then obtain an efficient random leader election protocol with reasonable fairness and also *perfect agreement* in the absence of private setups. The leader election protocol costs expected $\mathcal{O}(\lambda n^3)$ bits and expected constant asynchronous rounds, and can directly be plugged in all existing VBA protocols (i.e., multi-valued Byzantine agreement with external validity) [16,5,52,38,37] to replace its counterpart relying on private setups. The resulting VBA protocols can realize the maximal $n/3$ resilience and optimal

[3] Remark that there might exist efficient reduction from ABA to VBA in the public key infrastructure setting, which was discussed in [16]. Therefore, the recent private-setup free VBA protocol in [4] also improves ABA.

expected constant rounds, 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.

– Along the way, we develop a set of crucial techniques that could be of independent interests. We set forth a new primitive called weak core-set selection (WCS) to simplify the cumbersome component of information gather in CR93 [20] and AJM+21 [4]. Recall that information gather is a multi-sender extension of reliable broadcast [14], such that each party reliably broadcasts a value and then outputs a set of values that is a superset of some $(n-f)$-sized core-set. Selecting a core-set out of n broadcasted values requires another $2n$ reliable broadcasts in [20]. We conceptually weaken the primitive in a way that $f+1$ honest parties (instead of all honest parties) are ensured to output a superset of the core-set. This appropriate weakening significantly simplifies the protocol (i.e., replace a couple of reliable broadcasts by only two multicast rounds), and still it is a powerful building block, from which we can implement efficient common coin in the PKI setting.
We also give an efficient AVSS construction (satisfying the classic CR93 notion [20]) with only bulletin PKI setup (under the discrete logarithm assumption). The AVSS protocol is adaptively secure, and costs only $\mathcal{O}(n^2)$ messages and $\mathcal{O}(\lambda n^2)$ bits when sharing a λ-bit secret. Prior art with the same communication complexity either relies on private setup [8,42,43] or incurs at least $\mathcal{O}(\lambda n^3)$ bits [15,9] (except two recent work [6,62], yet they still have an extra $\log n$ factor).

1.2 Challenges and our techniques

Remaining efficiency hurdles. To flip coins, both CR93 and AJM+21 let each party commit an unbiased secret gathered from enough parties. In CR93, everyone plays a role of "delegate" for each party to share a secret through AVSS, then everyone picks and commits $n-f$ secrets from distinct "delegates", the aggregation of which is uniformly distributed. In AJM+21, each party again plays a role of "delegate" to choose a random secret for everyone, which is hidden in form of aggregatable public verifiable secret sharing (PVSS). Every party now combines $n-f$ PVSS scripts from distinct "delegates" to obtain an aggregated PVSS, which also hides an unbiased secret. Then, the aggregated PVSS script, which has $\mathcal{O}(\lambda n)$ bits, can be committed via a reliable broadcast (the broadcast of PVSS can be analog to AVSS, though no explicit AVSS invocation).

After enough parties commit their unbiased secrets, it invokes a procedure to select a core-set of these secrets. That means, all parties would choose a set of indices corresponding to some indeed committed (yet unknown) secrets, and more importantly, the honest parties' choices shall have a large enough intersection (called core-set). Hence, a simple trick to flip the coin can be imagined: each party just reconstructs the secrets, then the lowest bit of the largest reconstructed secret becomes the coin. This works because the largest secret has a constant probability to appear in the core-set. Usually, such a core-set is obtained via several reliable broadcasts [7,1,20], whose input is a set of $\mathcal{O}(n)$ indices.

As such, further reducing the communication of the CR93 and AJM+21 frameworks seems challenging, because every party reliably broadcasts and/or verifiably shares at least $\mathcal{O}(\lambda n)$ bits. While, on the other side, although KMS20 gives a workaround to the above steps and reduces the number of secrets to share by an $\mathcal{O}(n)$ order, it on the contrary causes slow termination of $\mathcal{O}(n)$ rounds.

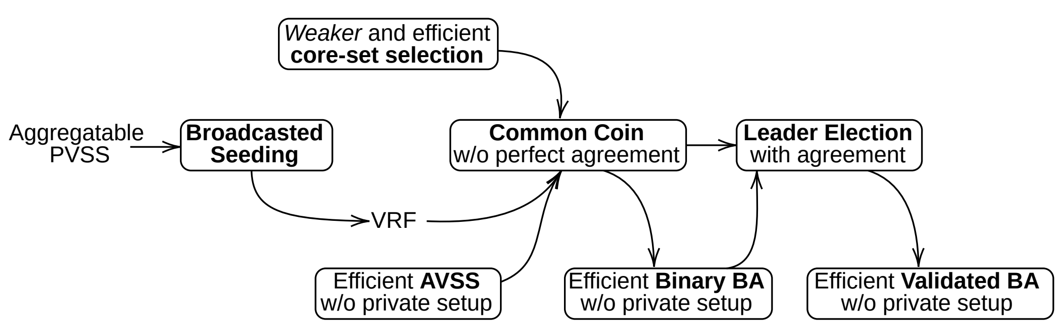

Fig. 1: Our new path to constructing efficient asynchronous common randomness and Byzantine agreement protocols in the absence of private setups.

Techniques to efficient common coin from PKI. We present *a collection of new techniques* to circumvent the efficiency hurdles lying in the phases of committing secrets and selecting core-set, as illustrated in Figure 1. In greater detail, we proceed as:

An efficient construction of private-setup free AVSS. The underlying AVSS shall cost at most quadratic communication, if we aim at common coin with cubic communication. We realize this by lifting Pedersen's verifiable secret sharing [59] to asynchronous network, through exploiting the wisdom of hybrid secret sharing [48]. First, the dealer just collects $n-f$ signatures from distinct parties for the same Pedersen's polynomial commitment [59]. Such that these signatures can ensure that at least $f+1$ honest parties receive the same commitment binding a unique encryption key. Then, the dealer uses the key to encrypt its actual secret, and leverages the $n-f$ solicited signatures to convince the honest parties to participate into a reliable broadcast for the ciphertext. By the design, it avoids reliably broadcasting the large polynomial commitment to the whole network.

Weakening Core-Set selection for more efficient construction. As earlier pointed out, the selection of core-set is important to harvest some reasonable probability to make the honest parties output some common randomness. And we further tackle the problem of how to efficiently attain core-set. Our main observation is that if the core-set primitive is used to pick some confidentially shared secrets that come with some "proofs" attesting the unbiased generation, e.g., VRFs, we might slightly weaken the primitive. I.e., only $f+1$ honest parties obtain the core-set, instead of all honest parties. This works because if the largest VRF appears in this weaker core-set, $f+1$ parties can reconstruct this largest VRF, and then multicast it to let all parties accept it. The weaker notion can be easily designed in the PKI setting: each party multicasts its input set, and then waits for $n-f$ signatures returned by distinct parties, indicating that at least $f+1$ honest parties have a superset. This becomes an efficient workaround to avoid $\mathcal{O}(n)$ reliable broadcasts in the conventional core-set selection [20,4,1,7].

A broadcast version "coin" to patch VRF. With efficient AVSS and core-set selection at hand, it is enticing to let every party confidentially share its own VRF evaluation with proof via AVSS (instead of gathering an unbiased secret from many parties). Then, the weak core-set selection can ensure that at least $f+1$ honest parties get the core of $n-f$ shared VRFs. As such, the largest VRF seemingly can appear in the core-set with a constant probability, and its least significant bits would naturally become the common coin.

However, the above seemingly appealing idea does not work in the bulletin PKI setting, because if the VRFs are evaluated on deterministic seeds. Corrupted parties can simply exploit malicious key generation to bias its own VRF, for example: just run key generation for polynomial times, choose the most favorable VRF key pair, and register the verification key at PKI. As such, VRFs evaluated by corrupted parties gain great advantage to be the largest. Thus, the adversary at least can nearly always know the flipped coin in advance.

To patch VRF with providing an unpredictable seed, we set forth a notion called reliable broadcasted seeding (Seeding) and construct it from aggregatable PVSS [40]. To some extent, Seeding can be viewed as a broadcast version common coin, which "broadcasts" an unpredictable VRF seed and is led by the party who evaluates the VRF. Moreover, as long as an honest party gets the VRF seed from the Seeding protocol, all honest parties would do so, and thus all parties can verify the leader's VRF evaluation on the seed. On the other side, if a corrupted Seeding leader intentionally blocks the protocol, no honest party can verify its VRF evaluation, and this actually "harm" the corrupted leader itself, because no honest party would support to solicit it into the core-set.

Putting everything together, we design a novel Coin framework, Then a weak core-set can gather $n-f$ AVSSes that share VRFs (patched by Seeding). With $1/3$ probability, the largest VRF appears in the core and is also evaluated by some honest party, and its lowest bits become the flipped coin.

Lifting agreement for Leader Election. Although common coin is a powerful tool to enable asynchronous Binary agreement, it faces a few barriers for implementing the interesting class of validated Byzantine agreements (VBA), thus missing the key step to reduce the communication complexity of expected constant-round ADKG [4]. The main issue stems from the fact that most existing VBAs [16,5,52] require a leader election with at least one significant strengthening relative to common coin, i.e., ensuring agreement all the time.

Noticing that our Coin protocol only lacks agreement in some unlucky cases (when the largest VRF does not appears in the core-set), we introduce a single ABA instance along with a set of voting rules to "detect" the possible disagreement, thus lifting Coin to attain perfect agreement.

In particular, everyone reliably broadcasts the speculative largest VRF heard at the end of Coin execution, and then waits for $n - f$ such broadcasted VRFs. If there exists a majority VRF (out of the $n - f$ received) that is also the largest, they vote 1 to ABA; otherwise, they vote 0. When ABA returns 0, it is okay for the honest parties to elect a default leader, e.g., the first party; while if ABA outputs 1, any two honest parties shall not have two distinct VRF evaluations that is both largest and majority, because the VRF evaluation satisfying the rules must be unique, so they can just select the leader according to the lower bits of this VRF evaluation. This fixes the lack of agreement in common coin without incurring extra factors in the asymptotic complexities.

1.3 Application scenarios

ABA and VBA protocols are at the core of many asynchronous fault-tolerant systems (e.g., BFT state-machine replication and robust MPC service) deployed over the unstable wide-area network [54,39,51]. Our techniques can remove unpleasant setup assumptions of them, and therefore have a broad array of applications. Here are a couple of typical examples:

Fast asynchronous DKG with cubic communication. Our new path to efficient VBA protocols can be used to replace the VBA instantiation in AJM+21 [4] to improve the same paper's asynchronous distributed key generation (ADKG) protocol, thus reducing the communication cost from $\mathcal{O}(\lambda n^3 \log n)$ to $\mathcal{O}(\lambda n^3)$ with preserving all other benefits such as fast termination in expected constant rounds and optimal resilience, cf. Section 7.3 for details.

Asynchronous random beacon without DKG. Our random leader election protocol can be easily adapted to realize randomness beacon, which can continually output a sequence of unbiased random strings in the asynchronous network. Here an unbiased random string means that its distribution is uniform (taken over all possible executions) despite the adversary. Conceptually, the construction only has to sequentially run many leader election protocols, and thus preserves expected $O(\lambda n^3)$ communication and $O(1)$ rounds. Different from prior asynchronous randomness beacon [17] that has to run ADKG [4,47,25] to bootstrap, our implementation does not go to ADKG, thus being more friendly for dynamic join and leave, cf. Section 7.3 for more details.

2 Other Related Work

Byzantine agreement was introduced by Shostak, Pease and Lamport [49]. Since then, it has been extensively studied in various settings with different flavors [34,32,30,2,28,56,26,44,13]. In the asynchronous network, the Byzantine agreement problem has to be solved via randomized protocols, and was initially studied by Ben-Or [11] and Rabin [60], respectively.

Kokoris-Kogias et al. [47] and Abraham et al. [4] also lifted their private-setup free Byzantine agreement protocols towards asynchronous distributed key generation (ADKG), thus emulating the trusted dealer of threshold cryptosystems against asynchronous adversary. The rationale behind ADKG is that it might amortize the cost of asynchronous protocols, if the threshold cryptosystem is used for many times. The latter study (AJM+21) is the state-of-the-art ADKG approach with using private-setup free VBA as the core building block. Our new path to VBA can directly reduce its communication cost by a $\log n$ factor.

Cohen et al. [22] constructed an asynchronous common coin with sub-optimal resilience using VRFs, but it uses pre-determined nonce for VRFs. So it actually takes an implicit assumption that either a trusted third-party performs honest VRF key generations on behalf of all parties or a common random string can be provided after all parties register at PKI. Gągol et al. [35] presented how to efficiently re-configuring asynchronous random beacon by running over permissioned directed acyclic graph (DAG). However, their bootstrapping might similarly need a common random string (generated after the registration of parties). We insist on more stringent setting without on-line common random strings. In addition, our leader election can be plugged in such DAG-based protocols [35,45] to make them efficient and private-setup free.

There are many AVSS protocols [8,42] that can realize optimal communication with relying on private setups. Prior to this paper and a concurrent work [25], the best known result are a couple of recent studies [6,62] that incurs $\mathcal{O}(n^2)$ messages and $\mathcal{O}(\lambda n^2 \log n)$ bits for λ-bit input secret. Some AVSS protocols [42,6] also focus on linear amortized communication for sufficiently large input secret, but they remain to exchange quadratic messages and bits while sharing a short secret. Our AVSS can easily combine the information dispersal technique [18] to realize the same linear amortized communication.

In the informational theoretic setting, a few interesting studies explored asynchronous leader election and byzantine agreement [41,46], but they realized sub-optimal resilience and non-constant asynchronous rounds with a security guarantee of only $1 - 1/poly(n)$. Our results and the prior art [4,18] in the computational setting can attain optimal $n/3$ resilience and (expected) constant asynchronous rounds, with overwhelming probability $1 - 1/superploy(n, \lambda)$, conditioned on the hardness of underlying cryptographic assumptions.

Common coin can also circumvent the Dolev-Strong bound [29] to fasten the termination of (partially) synchronous protocols [27,3,44,31]. Technique-wise, Afgjort [27] can be thought of our counterpart in the (partially) synchronous setting with the extra help from on-line common random strings, it also gathers VRFs in a core-set, and then uses the least significant bits of the maximal VRF to toss coins. However, Afgjort explicitly relies on the (partial) synchrony assumption to wait for that the largest VRF appears in the core-set. In contrast, we make two significant enhancements to adapt our private-setup free asynchronous setting. First, unpredictable seeds are generated on the fly to patch VRFs due to the lack of on-line common random string. Second, AVSS and a special asynchronous core-set selection protocols are designed to ensure that honest parties' VRFs are not leaked until a large enough core-set is fixed.

Note on results in a concurrent work [25]. A concurrent work from Das et al. [25] presented the technique of asynchronous data dissemination (ADD) to improve the efficiency of reliable broadcast and relevant protocols such as AVSS. It can reduce the communication of the specific AJM+21 VBA protocol to $\mathcal{O}(n^3)$ by replacing the reliable broadcast building block. It also has applications to reduce the communication of AJM+21 ADKG to cubic. Though Das et al.'s reliable broadcast and some of their proposed AVSS protocols can be adaptively secure, their applications to VBA and ADKG are also in the random oracle model against only static corruptions.

Different from Das et al. that focused on improving the broadcast components, we present a set of very different techniques to simplify the protocol structures of common coin and random leader election, which are the basis protocols of fast-terminating Byzantine agreement. In particular, our common coin and leader election can be directly plugged into any Byzantine agreement protocols that requires such a building block to improve the efficiency of their private-setup free variants, while ADD only explicitly helps the specific AJM+21 VBA (e.g., if without our results or future studies on asynchronous common randomness). In addition, our leader election protocol can be adapted into a reconfiguration-friendly random beacon protocol with DKG, while Das et al.'s results can only bootstrap random beacon protocol through DKG. Moreover, it might be interesting to explore the new design space provided by the combination of ADD and our techniques towards practical private-setup free asynchronous protocols.

3 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 fully-meshed asynchronous message-passing model with Byzantine corruptions and bulletin public key infrastructure (PKI). 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. Once a public key is registered, we assume all parties can receive them immediately from the PKI.

- *Computing model.* Following [16,5] and modern cryptographic practices, 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 $n/3$ Byzantine corruptions.* The adversary can choose up to f out of n parties to corrupt and fully control. No asynchronous BA protocols can tolerate more than $f = \lfloor (n-1)/3 \rfloor$ such Byzantine corruptions, so this is the 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 while registering public keys at 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 it is computationally infeasible for the adversary to 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. Here we assume asynchronous secure channels (instead of merely reliable asynchronous channels) for presentation simplicity, and they can be obtained from the bulletin PKI through authenticated key exchange, and therefore are not extra assumptions.
- *Miscellany.* All system parameters, such as n, are (probably unfixed) polynomials in the security parameter λ [16,5,52].

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) under the influence of adversary.
- *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.* The eventual delivery of asynchronous network 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 [20]. We then can measure the running time by counting such asynchronous "rounds".

Note on "private-setup free". More precisely, our private-setup free model admits bulletin PKI with some system parameters that are just group descriptions and random group generators. These parameters are indeed one-time setup (belonging to a global system instead of just for ours), and are usually ignored in the literature [4,25]. Except that, we consider other structured common reference strings such as that of KZG polynomial commitment [43] fall into the category of private setups. We might consider that no one can stay on-line to provide a trusted common random string after PKI registration, which is the subtle reason why we have to patch VRF with a broadcasted seeding protocol.

Note on static/adaptive adversary. Some of our results (e.g., our AVSS protocol) can be secure against an adaptive adversary that can corrupt up to f parties while the protocol is running. While our common coin and random leader election protocols are secure in the static model, in which the adversary is restricted to corrupt parties before the protocol starts. However, this is only because the existing aggregatable PVSS scheme is not proven to be adaptively secure (which actually is the same reason why AJM+21 [4] is statically secure). To demonstrate that, we can introduce a one-time online common random string assumption, thus avoid the broadcasted seeding protocol that relies on PVSS, and then show that our common coin (as well as random leader election) become adaptively secure. Namely, we can assume a trusted one-time randomness that is announced after PKI registration but before protocol execution, and adaptive security can be realized by our protocols in the setting, as PVSS is no longer needed, cf. more detailed discussions in Section 6.

Moreover, the assumption of adaptively secure private channels can be easily realized by existing techniques (e.g., in the erasure model [19]).

4 Preliminaries

Reliable broadcast (RBC) [14] is a protocol among a set of n parties, in which a party called sender aims to send a value to all. It satisfies the next properties:

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

We require the digital signature scheme to be existentially unforgeable under an adaptive chosen-message attack (i.e., EUF-CMA secure). In the bulletin PKI setting, every party is bounded to a unique verification key for signature. For presentation brevity, in a protocol with an explicit identifier ID, we might let $\mathsf{Sign}_i^{\mathsf{ID}}(m)$ 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, 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 certain party $\mathcal{P}_i$.

Verifiable random function. A verifiable random function (VRF) [53] is a pseudorandom function that also returns a proof to attest that the correctness of its evaluation result. 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.

VRF shall satisfy *unpredictability*, *verifiability* and *uniqueness*. Here *verifiability* conventionally 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 computationally 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 computationally infeasible to find x, r_1, r_2, π_1, π_2 such that $r_1 \neq r_2$ but $\mathsf{VRF.Verify}(pk, x, r_1, \pi_1) = \mathsf{VRF.Verify}(pk, x, r_2, \pi_2) = 1$.

Taking the bulletin PKI for granted, everyone can be associated to a unique VRF public key. For example, an honest party $\mathcal{P}_i$ runs VRF.Gen to generate a unique pair of private key sk_i and public key pk_i, and then registers its pk_i via the PKI. However, traditional VRF notion [53] does not malicious key generation done by corrupted parties. To capture such threat, we actually require a stronger *unpredictability* property called *unpredictability under malicious key generation* due to David et al. [26] throughout 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 can be achieved in the random oracle (RO) model under CDH assumption [26].

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. Similarly, $\mathsf{VRF.Verify}_i^{\mathsf{ID}}(x, r, \pi)$ then denotes $\mathsf{VRF.Verify}(pk_i, \langle \mathsf{ID}, x \rangle, r, \pi)$.

(Aggregatable) public verifiable secret sharing. A (n, t) non-interactive PVSS scheme can be described as a tuple of non-interactive algorithms as follows (with taking param as an implicit input):

- $\mathsf{Deal}(\mathsf{ek}, s) \rightarrow \mathsf{pvss}$ is an algorithm that takes a secret s as input and outputs a script pvss.
- $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{pvss}) \rightarrow 0/1$ is a deterministic algorithm that takes all encryption keys as input, and 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).
- $\mathsf{GetShare}(dk_i, \mathsf{pvss}) \rightarrow \mathsf{sh}_i$ is executed by the party $\mathcal{P}_i$, takes a valid pvss script and $\mathcal{P}_i$'s decryption key dk_i as input, and outputs the secret share sh_i of the secret committed to pvss.
- $\mathsf{VrfyShare}(j, \mathsf{sh}_j, \mathsf{pvss}) \rightarrow 0/1$ takes the PVSS script pvss and party $\mathcal{P}_j$'s secret share sh_j as input, and verifies whether sh_j is the correct j^{th} share of the polynomial committed to pvss or not.
- $\mathsf{AggShares}(\{(j, \mathsf{sh}_j)\}_t) \rightarrow a$ takes t valid secret shares from distinct parties regarding an implicit PVSS script pvss, and computes the secret a committed to the pvss.
- $\mathsf{VrfySecret}(s, \mathsf{pvss}) \rightarrow 0/1$ verifies whether a secret s is indeed committed to pvss or not.

Gurkan et al. [40] recently proposed to lift PVSS scheme to further enjoy aggregability, which need to slightly adapt the syntax. Here we only highlight the small adaptions to these algorithmic interfaces:

- $\mathsf{Deal}(\mathsf{ek}, sk_i, s) \rightarrow \mathsf{pvss}$. Now the algorithm takes an extra secret signing key sk_i as input, which is needed to make the pvss script to carry an unforgeable weight tag bounded to the identity $\mathcal{P}_i$.
- $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{vk}, \mathsf{pvss}) \rightarrow 0/1$. It takes some verification keys vk besides ek and pvss as input. The output still represents whether pvss is valid or not.
- $\mathsf{AggScripts}(\mathsf{pvss}_1, \mathsf{pvss}_2) \rightarrow \mathsf{pvss}$. This is a newly introduced algorithm that takes two valid PVSS scripts pvss_1 and pvss_2 as input and outputs a valid PVSS script pvss.
- $\mathsf{Weights}(\mathsf{pvss}) \rightarrow \boldsymbol{w}$. This is another new algorithm. It takes a valid pvss script as input and outputs an n-sized vector $\boldsymbol{w}$, 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 aggregatable PVSS scheme due to Gurkan et al. [40] satisfies a few nice security properties such as *verifiable commitment*, *verifiable aggregation* and *secrecy*. Informally, *verifiable commitment* means that any party can verify that a PVSS script pvss indeed commits a fixed secret s that can later be collectively reconstructed by the participating parties; *secrecy* means that it is infeasible for an adversary to compute the committed secret from the PVSS script; *verifiable aggregation* means if Weights(pvss) returns $(w_1, w_2, \cdots, w_n)$, then the secret s committed to pvss indeed equals $\sum_{i=1}^{n} w_i s_i$, where s_i is the secret committed to some PVSS script pvss_i that is solely generated (and signed) by the party $\mathcal{P}_i$. We defer the detailed descriptions of these properties to Appendix B.

5 Warm-up: AVSS and Weaker Core Set from PKI

As briefly mentioned in Introduction, our common coin and leader election protocols require a more efficient private-setup free AVSS instantiation and an efficient construction for a weaker core-set selection (WCS) notion. When construing our common coin, AVSS is used to let everyone confidentially share an unbiased VRF evaluation, and WCS can be used let enough honest parties hold an intersecting core-set containing at least $n - f$ completed AVSSes. Thus, with a constant probability, the largest VRF evaluated by some honest party can appear in the core-set, and its lowest bits are ensured to be the common coin.

This Section focuses on the needed preparing building blocks — AVSS and core-set selection. The AVSS protocol to present attains $\mathcal{O}(\lambda n^2)$ bits in the PKI setting.[4] Then, we put forth to and construct a weak core-set selection, which can ensure $f + 1$ honest parties (instead of all) to get some superset of a $(n - f)$-sized core-set.

[4] Through the paper, the input secret to AVSS is assumed small, e.g., $\mathcal{O}(\lambda)$ bits.

5.1 Efficient Private-Setup Free AVSS

Instead of varieties of strengthened AVSSes, we focus on the hereinbelow classic AVSS notion defined by Canetti and Rabin in 1993 [20].

Definition 1 (Asynchronous Verifiable Secret Sharing [20]). *An* AVSS *consisting of a tuple of protocols* (AVSS-Sh, AVSS-Rec) *can be defined as follows.*

Syntax. *In each* AVSS-Sh *instance with a session identifier* ID*, a designated dealer* $\mathcal{P}_D$ *inputs a secret and each party outputs a string (e.g., a share of the input secret). In the corresponding* AVSS-Rec *instance, the parties input their outputs of* AVSS-Sh *to collectively reconstruct the shared secret.*

Properties. AVSS *satisfies next properties except with negligible probability:*

- **Totality**. *If some honest party outputs 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**. *When an honest party outputs in the* AVSS-Sh *instance for* ID*, there exists a fixed value* m^**, such that when all honest parties are activated to run the corresponding* AVSS-Rec *instance, all 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;*
  - *If any honest party reconstructs some value* m^* *in the corresponding* AVSS-Rec *instance,* $m^* = m$*.*
- **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), before the first honest party starts the corresponding* AVSS-Sh *instance. This can be formalized as that the adversary has negligible advantage in the Secrecy game (deferred to Appendix A).*

High-level rationale. The intuitions behind our AVSS construction is simple. Inspired by the hybrid approach of secret sharing [48], our sharing sub-protocol is split into two steps: (i) it first takes the advantage of PKI model to let the dealer multicast a polynomial commitment to an encryption key and then collect enough signatures (e.g., $n-f$) on the commitment, thus ensuing at least $f+1$ honest parties commit the same encryption key, and (ii) then the dealer multicasts the $n-f$ signatures solicited from the first step to convince the whole network to participate into a reliable broadcast to disseminate the ciphertext encrypting the actual secret.

While in the reconstruction phase, probably only $f+1$ honest parties might output the polynomial commitment to the decryption/encryption key, though all honest parties have the ciphertext. This seemingly causes some parties fail to reconstruct the secret (because they cannot decrypt). Nevertheless, the $f+1$ honest parties holding the correct commitment can recover the decryption key and then multicast it to all parties. So all honest parties can count whether the same key is from $f+1$ distinct parties, and finally use it to decrypt the secret.

Constructing AVSS without private setups. The rationale behind our AVSS construction is straightforward. The sharing phase splits the hybrid approach of secret sharing [48] into two steps: (i) it first takes the advantage of PKI model to let the dealer collect enough signatures (e.g., $n-f$) on a polynomial commitment [59] to an encryption key, thus ensuing at least $f+1$ honest parties commit the same encryption key, and (ii) then the dealer multicasts the $n-f$ signatures solicited from the first step to convince the whole network to participate into a reliable broadcast to disseminate the ciphertext encrypting the actual secret.

In particular, the sharing protocol AVSS-Sh proceeds in the following steps:

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 [59]. The dealer randomly constructs two polynomials $A(x)$ and $B(x)$ of degree at most f. Let $A(0) = key$. Then, the dealer computes a commitment $C = \{c_j\}$ to the polynomial $A(x)$ with using $B(x)$ for hiding, 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 $B(j)$.

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 $j \in [0, f]$,

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 KeyStored(ID, σ_j) from $\mathcal{P}_j$ s.t. $\mathsf{SigVerify}_j^{\mathsf{ID}}(C, \sigma_j) = 1$ **do**

8: $\Pi \leftarrow \Pi \cup \{(j, \sigma_j)\}$

9: **if** $|\Pi| = n - f$ **then**

10: $cipher \leftarrow key \oplus m$ and **multicast** Cipher($\mathsf{ID}, \Pi, C, cipher$) to all parties

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

11: $\mathsf{sh}_A \leftarrow \perp$, $\mathsf{sh}_B \leftarrow \perp$, $\mathsf{cmt} \leftarrow \perp$

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', $\sigma \leftarrow \mathsf{Sign}_i^{\mathsf{ID}}(C')$, **send** KeyStored($\mathsf{ID}, \sigma$) to $\mathcal{P}_D$

16: **upon** receiving Cipher($\mathsf{ID}, \Pi, C, cipher$) from $\mathcal{P}_D$ for the first time **do**

17: **wait** for a valid KeyShare message s.t. $A'(i)$, $B'(i)$ and C' are recorded

18: **if** $C' = C$ and Π has $n - f$ valid signatures for C from distinct parties **then**

19: $\mathsf{sh}_A \leftarrow A'(i)$, $\mathsf{sh}_B \leftarrow B'(i)$ and $\mathsf{cmt} \leftarrow C'$

20: **multicast** Echo($\mathsf{ID}, cipher$) to all parties

21: **upon** receiving $2f + 1$ Echo($\mathsf{ID}, cipher$) from distinct parties **do**

22: **multicast** Ready($\mathsf{ID}, cipher$) to all parties if Ready not sent yet

23: **upon** receiving $f + 1$ Ready($\mathsf{ID}, cipher$) from distinct parties **do**

24: **multicast** Ready($\mathsf{ID}, cipher$) to all parties if Ready not sent yet

25: **upon** receiving $2f + 1$ Ready(ID, c) from distinct parties **do**

26: **output** $(cipher, \mathsf{sh}_A, \mathsf{sh}_B, \mathsf{cmt})$

Once a party $\mathcal{P}_i$ receives KeyShare message from the dealer, it checks that C indeed commits $A(i)$ with using $B(i)$ for blinding, and returns a signature for C to the dealer via a KeyStored message.

2. *Cipher broadcast* (Line 7-10, 16-26). After receiving $n-f$ valid KeyStored messages from distinct parties, the dealer sends a Cipher message to all parties containing a ciphertext *cipher* encrypting its input m, the commitment C, and a quorum proof Π containing $n - f$ valid signatures for C. The remaining process of the phase is similar to a Bracha's reliable broadcast for *cipher* [14], except that a party would not "echo" *cipher* if not yet receiving valid Π for C. At the end of the phase, each party can output $(cipher, A(i), B(i), C)$, where $A(i)$, $B(i)$ and C can be $\perp$.

Then, the AVSS-Rec phase can be activated to proceed in the next two phases:

1. *Key recovery* (Line 1-10). For each party $\mathcal{P}_i$ that activates AVSS-Rec with taking the output of AVSS-Sh as input, it sends KeyRec message containing $A(i)$ and $B(i)$, in case these variables are not $\perp$. At least $f + 1$ honest parties have already received the commitment C, so they can eventually solicit $f+1$ valid shares of the polynomial committed to C through KeyRec messages, and then interpolate the shares to reconstruct $A(x)$ and compute $key = A(0)$.
2. *Key amplification.* After a party obtains decryption *key*, it multicasts *key* via a Key message. So all honest parties can receive $f + 1$ Key messages containing the same *key*, and then compute and output $m = key \oplus cipher$.

For the sake of completeness, we also present formal pseudocode descriptions for AVSS-Sh and AVSS-Rec in Alg. 1 and Alg. 2, respectively.

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

Initialization: $\Phi \leftarrow \emptyset$

upon being activated with input $(cipher, \mathsf{sh}_A, \mathsf{sh}_B, \mathsf{cmt})$ **do**
  if $\mathsf{cmt} \neq \perp$, $\mathsf{sh}_A \neq \perp$ and $\mathsf{sh}_B \neq \perp$ **then**
    multicast $\text{KeyRec}(\mathsf{sh}_A, \mathsf{sh}_B)$ to all parties

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

upon receiving $f+1$ Key messages containing the same key **do**
  $m \leftarrow key \oplus cipher$ and **output** m

Security analysis of AVSS. The intuition of proving our 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 [59] about his verifiable secret sharing; the commitment is ensured by Pedersen commitment and the unforgeability of signatures. More formally, the security of our AVSS protocol can be proved as follows.

Lemma 1. *If any two honest parties $\mathcal{P}_i$ and $\mathcal{P}_j$ output $(cipher, \cdot, \cdot, \cdot)$ and $(cipher', \cdot, \cdot, \cdot)$ in AVSS-Sh[ID], respectively, then $cipher = cipher'$ except with negligible probability.*

Proof. Suppose that $cipher \neq cipher'$, $\mathcal{P}_i$ receives $2f+1$ Ready messages containing $cipher$, 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 $cipher'$ 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 $cipher \neq cipher'$, implying $cipher = cipher'$.

Lemma 2. *If some honest party outputs 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.*

Proof. 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.

Lemma 3. *When some honest party outputs in the AVSS-Sh instance for ID, there exists a value m^* that is fixed associated to ID.*

Proof. Firstly, we prove that if any two honest parties record cmt and cmt' respectively, then $\mathsf{cmt} = \mathsf{cmt}'$. Note that honest party will record a cmt only if it receives a valid Π containing $n-f$ valid signatures for cmt. Suppose that $\mathsf{cmt} \neq \mathsf{cmt}'$, $\mathcal{P}_i$ receives a Π, which means at least $f+1$ honest parties have signed for cmt; in the same way, $\mathcal{P}_j$ receives a Π' and at least $f+1$ honest parties have signed for cmt'. So according to the unforgeability of digital signatures, at least one honest party signed for both cmt and cmt', which is impossible. Thus, $\mathsf{cmt} = \mathsf{cmt}'$. 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^*. From Lemma 1 and the totality of AVSS, when some honest party outputs in the AVSS-Sh instance for ID, all honest parties receive the same cipher $cipher$. So there exists a unique $m^* = cipher \oplus key^*$, which can be fixed once some honest party outputs in AVSS-Sh.

Lemma 4. *When all honest parties activate* $\mathsf{AVSS\text{-}Rec}$ *on* ID*, each of them can reconstruct the same value* m^*

Proof. Any honest party outputs in the $\mathsf{AVSS\text{-}Sh}$ 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 Π. Since the signatures in Π are unforgeable, 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 $\mathsf{AVSS\text{-}Rec}$, 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 $m^* = cipher \oplus key^*$ as they obtain the same ciphertext *cipher* from $\mathsf{AVSS\text{-}Sh}$.

Lemma 5. *If the dealer is honest and all honest parties are activated to run* $\mathsf{AVSS\text{-}Sh}$ *on* ID*, all honest parties would output in the* $\mathsf{AVSS\text{-}Sh}$ *instance.*

Proof. If the dealer is honest and all honest parties are activated, it is clear that (i) all honest parties can eventually wait the shares of $A(x)$ and $B(x)$ as well as the same commitment C so all honest parties will sign for C. Thus, the honest dealer must can collect at least $n-f$ valid digital signature for C from distinct parties to form valid Π and (ii) all honest parties can eventually broadcast the same ECHO messages and the same READY messages after receiving the shares of $A(x)$ and $B(x)$ as well as the same C, thus finally outputting in the $\mathsf{AVSS\text{-}Sh}$ instance.

Lemma 6. *If the dealer is honest and inputs secret* m*, the value* m^* *reconstructed by any honest party in the corresponding* $\mathsf{AVSS\text{-}Rec}$ *instance must be equal to* m*, for all* ID*.*

Proof. From Lemma 4, we have proved that all honest parties will reconstruct the m^* which is fixed when some honest party completes the $\mathsf{AVSS\text{-}Sh}$. So all we need is to prove that the fixed m^* is equal to the m that the honest dealer inputs in the $\mathsf{AVSS\text{-}Sh}$. It is easy to see that (i) any honest party must output a ciphertext *cipher* same to the ciphertext computed by the honest sender and (ii) due to the correctness and binding of commitment scheme honest parties must receive the same C to $A(x)$ and $B(x)$, where $A(x)$ and $B(x)$ are chosen by the honest deader. So $m^* = cipher \oplus A(0) = m$.

Lemma 7. *In any* $\mathsf{AVSS\text{-}Sh}$ *instance, if the dealer is honest, the adversary shall not learn any information about the key shared by the dealer from its view.*

Proof. The adversary's *view* in an $\mathsf{AVSS\text{-}Sh}$ execution with an honest dealer would include the commitment C, the ciphertext *cipher*, some signatures for C, 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 shared *key* (even if the adversary can fully break digital signature to learn the private signing keys). Thus, following the information-theoretic argument in [59], since the commitment C is perfectly hiding and f shares of Shamir's secret sharing scheme also leaks nothing about the key, the adversary can learn nothing about *key*.

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* $n/3$ *adaptive byzantine corruption and bulletin PKI assumption without private setups, conditioned on the hardness of Discrete Log problem and* $\mathsf{EUF\text{-}CMA}$ *security of digital signature.*

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

– *Totality.* Totality can be proved immediately from Lemma 2.
– *Commitment.* Commitment can be proved from Lemma 3 and Lemma 4.
– *Correctness.* Correctness can be proved from Lemma 5 and Lemma 6

– *Secrecy.* From Lemma 7, if the dealer is honest, the adversary can learn nothing about key. So the adversary cannot distinguish the distribution of $cipher = key \oplus m_b$ and a uniform distribution (otherwise, the adversary can be invoked to break Lemma 7). Therefore, the adversary's advantage in the Secrecy game $\mathbf{Adv_{sec}}$ is negligible.

Remark on adaptive security. Our AVSS protocol is adaptively secure. This is because our AVSS-Sh sub-protocol is similar to that of [15], which use Pedersen's polynomial commitment in combination of Shamir's secret sharing to consistently distribute the secret shares (of an encryption key) to the participating parties, which avoids the shortage of using static cryptographic primitive such as non-interactive PVSS. While the major difference between [15] and our AVSS-Sh is that we concatenate $n - f$ EUF-CMA secure digital signatures to form a quorum proof attesting that enough honest parties have received the consistent secret shares. Nevertheless, this doesn't sacrifice adaptive security, because the quorum proof ensures that there must exist $f+1$ honest parties that can never be corrupted and also received the consistent secret shares, and hence these $f + 1$ forever honest parties can help all other honest parties to recover the same encryption key during the AVSS-Rec protocol.

Complexities of AVSS. The complexities of the AVSS protocol shown in Alg. 1 and Alg. 2 can be easily seen: Both AVSS-Sh and AVSS-Rec protocols cost at most a constant number of asynchronous rounds to terminate. Each round at most exchanges n^2 messages, indicating $O(n^2)$ message complexity. Moreover, there are $O(n)$ messages having $\mathcal{O}(\lambda n)$ bits and $O(n^2)$ messages having $\mathcal{O}(\lambda)$ bits, thus the communication complexity of the protocol is of overall $O(\lambda n^2)$ bits. Recall that λ captures the size of cryptographic objects.

5.2 Weak Core Set Selection

Core-set selection is a critical component while flipping a coin in the asynchronous network [20,4]. It allows each party to output a set of indices representing some completed reliable broadcasts [4] or completed AVSSes [20], and more importantly, the the intersection of all honest parties' outputs corresponds to an always large enough core-set (e.g., $n - f$).

Instead of this widely known approach, here we introduce a weakened core set selection primitive in which probably only $f + 1$ honest parties can receive the core. In the presence of PKI, it can be constructed very efficiently, and remains to be an expressive notion while flipping a coin. The idea is to use it select a core-set of AVSSes that hide some VRFs. With a constant probability, the largest VRF can luckily appear in the core-set, so at least $f + 1$ honest parties can reconstruct this largest VRF and multicast it to the whole network, thus still ensuring all honest parties to get the largest VRF.

Let us focus on this weakened notion and its efficient construction. More formally, we can define the weak core-set notion as follows.

Definition 2 (Weak Core-Set Selection). *A protocol among n parties with up to f Byzantine corruptions realizes a weak core-set selection, if has syntax and properties as follows.*

Syntax. *For each protocol instance with session identifier* ID*, every party $\mathcal{P}_i$ inputs a set of indices S_i s.t. $|S_i| \geq n - f$. Note that each index belongs to $[n]$, and each honest party's input set S_i can monotone increase over the protocol execution. Then, every honest party $\mathcal{P}_i$ outputs a set of indices $\hat{S}_i$.*

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

– **Termination**. *If any index in any honest party's input set can eventually appear in all honest parties' input sets, then every honest party would output.*
– $(f + 1)$**-Supporting Core-Set**. *Once the first honest party outputs, there exists a core-set S^* consisting of at least $n - f$ distinct indices, and S^* must be the intersection of at least $f + 1$ honest parties' output sets.*
– **Validity**. *Any index in the honest parties' outputs can be found in some honest party's input set.*

Intuitively, the above definition captures our purpose that after each party conducts reliable broadcast or verifiable secret sharing, each party can invoke the primitive to output a set of indices representing which reliable broadcasts or AVSSes are indeed completed. More importantly, the output sets

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

$\widetilde{S} \leftarrow \perp$, $\hat{S} \leftarrow \perp$

upon receiving the input set S and $|S| \geq n - f$ **do**
  $\widetilde{S} \leftarrow S$ and **multicast** $\text{Lock}(\mathsf{ID}, \widetilde{S})$ to all parties

upon receiving $\text{Lock}(\mathsf{ID}, \widetilde{S}_j)$ from $\mathcal{P}_j$ for the first time **do**
  if $|\widetilde{S}_j| \geq n - f$ **then**
    wait for $\widetilde{S}_j \subseteq S$
      $\sigma_i^j \leftarrow \mathsf{Sign}_i^{\mathsf{ID}}(\widetilde{S}_j)$ and **send** $\text{Confirm}(\mathsf{ID}, \sigma_i^j)$ to $\mathcal{P}_j$

upon receiving $\text{Confirm}(\mathsf{ID}, \sigma_j^i)$ from $\mathcal{P}_j$ s.t. $\mathsf{SigVerify}_j^{\mathsf{ID}}(\widetilde{S}, \sigma_j^i) = 1$ **do**
  $\Sigma \leftarrow \Sigma \cup \{j, \sigma_j^i\}$
  if $|\Sigma| = n - f$ **then**
    multicast $\text{Commit}(\mathsf{ID}, \Sigma, \widetilde{S})$ to all parties

upon receiving $\text{Commit}(\mathsf{ID}, \Sigma_j, \widetilde{S}_j)$ message from $\mathcal{P}_j$ for the first time **do**
  if Σ_j contains $n - f$ valid signatures for $\widetilde{S}_j$ from distinct parties **then**
    ▷ I.e., check $|\{k \mid (k, \sigma_k^j) \in \Sigma_j\}| = n - f \wedge \forall (k, \sigma_k^j) \in \Sigma_j$, $\mathsf{SigVerify}_k^{\mathsf{ID}}(\widetilde{S}_j, \sigma_k^j) = 1$
    $\widehat{S} \leftarrow S$ and **output** $\widehat{S}$

of at least $f+1$ honest parties share a $(n-f)$-sized intersection, representing that all these honest parties have output in these reliable broadcasts or AVSSes.

Constructing WCS without private setup. Here we present a concise construction of weak core set WCS (formally shown in Alg. 3):

1. Once an honest party $\mathcal{P}_i$ receives an input local set S which contains $n - f$ values, it takes a "snapshot" $\widetilde{S}$ of S and multicasts $\widetilde{S}$ to all parties. Note that $\mathcal{P}_i$'s local S can increase monotonically after the multicast, as new indices might be added to S. Then, if receiving some $\widetilde{S}_j$ sent from some party $\mathcal{P}_j$, the party $\mathcal{P}_i$ checks $|\widetilde{S}_j| = n - f$, and waits for that its local S eventually becomes a superset of $\widetilde{S}_j$, after which, it returns a signature for $\widetilde{S}_j$ to $\mathcal{P}_j$.
2. Eventually, $\mathcal{P}_i$ might collect $n - f$ distinct signatures for its multicasted "snapshot" $\widetilde{S}$, which corresponds to a quorum proof Σ for $\widetilde{S}$. Finally, $\mathcal{P}_i$ multicasts Σ and $\widetilde{S}$ to all parties. After receiving a valid quorum proof Σ_j for $\widetilde{S}_j$ from some party $\mathcal{P}_j$, the party $\mathcal{P}_i$ can immediately output its current local set S (without halt).

Security analysis of WCS. The security intuition of our WCS construction is that when the first honest party outputs, it must receive a valid quorum proof attesting that at least $f+1$ honest parties have signed the same set consisting of $(n-f)$ indices. Thus, these $f+1$ honest parties must have a $(n-f)$-sized intersection in their outputs. More formally, we can prove the following security theorem.

Theorem 2. *The algorithm shown in Alg. 3 realizes* WCS *against* $n/3$ *adaptive byzantine corruptions in the asynchronous message-passing model, conditioned on that the underlying digital signature scheme is* EUF-CMA *secure.*

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

– *Termination.* If any value v in some honest party's input set will eventually be included into all honest parties' input sets, any honest $\mathcal{P}_i$'s $\widetilde{S}_i$ will be included in all honest parties' local sets. So any honest $\mathcal{P}_i$ can collect a set Σ_i containing at least $n - f$ signatures for its $\widetilde{S}_i$ and multicast it to all parties via a Commit message. For any honest party $\mathcal{P}_i$, once it receives a valid Σ_j for the first time, it will fix a $\hat{S}$ and output.
– *(f+1)-Supporting Core-Set.* When the first honest party outputs from the protocol, it has received a valid Σ_j with $n - f$ signatures for S_j and 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 $S_j \geq n - f$ and $S_j \subseteq S_i$. Thus, trivially from the unforgeability of digital signatures, with all but negligible probability, once the first

honest party receives a valid Σ_j for S_j and outputs, there exists a core set $S^* = S_j$ which is subset of at least $f+1$ forever honest parties' local S. After that, if some of the $f+1$ forever honest parties can output a set $\hat{S}$, then $S^* \subseteq \hat{S}$.
- *Validity.* The validity of WCS is trivial since each honest party outputs its local set S which is the input of itself.

Remark on adaptive security. We concatenate $n-f$ EUF-CMA secure signatures from distinct parties to attest the existence of a core set, and therefore, whenever any so-far honest party outputs, there must exist $f+1$ honest parties that have signed the core set and can never be corrupted by the adaptive adversary, and at least these $f+1$ forever honest parties would share a $(n-f)$-sized intersection in their output sets. This facts prevent the adaptive adversary from corrupting parties posteriorly to make the core-set is received by less than $f+1$ honest parties.

Complexities of WCS. The complexities of the WCS protocol can be easily seen. All parties can terminate after three asynchronous rounds (i.e., Lock, Confirm and Commit). The over message complexity is $O(n^2)$, because each honest party sends at most $3n$ messages. Each message contains at most $O(\lambda n)$ bits, so the overall communication cost is $O(\lambda n^3)$.

6 Backbone: Reasonably Fair Common Coin from PKI

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 (Coin) which can be instantiated by AVSS, WCS along with using VRFs in the bulletin PKI setting. The Coin protocol attains constant running time, $\mathcal{O}(n^3)$ messages and $\mathcal{O}(\lambda n^3)$ bits. Thus, many existing ABA protocols [23,55] can directly adopt it for reducing private setup, and preserve other benefits such as expected constant rounds and optimal resilience, with incurring expected cubic communicated bits.

6.1 Common coin without private setup

The backbone of our results is an efficient private-setup free common coin (Coin) protocol that costs only $\mathcal{O}(\lambda n^3)$ communicated bits and terminate in constant rounds. Formally, we consider the common coin notion defined as follows:

Definition 3 **($(n, f, f+k, \alpha)$-Common Coin).** *A protocol realizes $(n, f, f+k, \beta)$-Coin, if it is executed among n parties with up to f static byzantine corruptions and has syntax and properties as follows.*

Syntax. *For all executions of 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.*
- **Reasonably fair bit-tossing**. *Prior to that k honest parties ($1 \le k \le f+1$) are activated on ID, the adversary $\mathcal{A}$ cannot fully guess the output. More precisely, consider the 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 say that $\mathcal{A}$ wins; we require $\Pr[\mathcal{A} \text{ wins}] \le 1-\alpha/2$.*
  Here α represents the lower-bound probability that all honest parties would output the same bit that is as if uniformly distributed over $\{0,1\}$, while $1-\alpha$ captures the possibility that the adversary might predicate/bias the output (which also capture the case that the honest parties output differently).

Intuitively, with at least α probability taken over all possible Coin executions, the adversary cannot predict the output bit better than guessing. A Coin protocol is said to be perfect, if $\alpha = 1$. Nevertheless, many ABA constructions [20,55] actually do not necessarily need perfect Coin, and can terminated in expected constant rounds with optimal $n/3$ resilience, as long as using a $(n, n/3, n/3+k, \alpha)$-Coin scheme, where $\alpha < 1$ is a certain constant. This is mainly because such constructions can tolerate the probable disagreement of such imperfect coins, and then repeat by iterations to explore the α-probability good case for terminating.

Tackling the seed of VRF. As briefly mentioned in Introduction, our Coin construction relies on VRF to let each party evaluate an unbiased random output, thus reducing the number of needed AVSSes. 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 [22], we aim to opt out of such private trusted setup for VRFs. So the VRF key generation is conducted by each participating party itself. That means, if the Coin protocol also uses some deterministic seeds for VRF evaluations, a compromised party can register at PKI with some maliciously chosen VRF keys, and probably can bias the distribution of its VRF during the protocol execution.

Trusted nonce from "genesis". In many settings, this might not be an issue, since there could be a trusted nonce generated after all parties have registered their VRF keys (e.g., the same rationale behind the "genesis block" in some Proof-of-Stake blockchains [26]), which can be naturally used as the VRF seed. Such a functionality was earlier formalized by David, Gazi, Kiayias and Russell in [26] as an initialization functionality to output an unpredictable VRF seed.

Generating VRF seed on the fly. Nevertheless, we might still expect less setup assumptions to get rid of the trusted "genesis block". 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 *reliable broadcasted seeding* (Seeding) as amendment to patch VRF with unpredictable nonce (or called seed interchangeably). Intuitively, the notion can be understood as a "broadcast" version of common coins, and might not terminate if encountering malicious leader. Seeding is a protocol with two successive committing and revealing phases, and can be defined as follows.

Definition 4 (Reliable Broadcasted Seeding). Seeding *is a protocol with two successive committing and revealing phases, and can be defined as follows.*

Syntax. *For each protocol instance with an identifier* ID*, it has a designated party called leader* $\mathcal{P}_L$ *and is executed among* n *parties with up to* f *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 outputs 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.
- **Committing**. *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* ($1 \le k \le f+1$) *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* k *honest parties are activated on* ID*, then the probability that* $seed^* = seed$ *shall be negligible, where seed is the output of some honest party for* ID.

Remark on committing and unpredictability. Combining the committing and unpredictability properties would ensure that no one can predicate the output, before the seed to output is already fixed, which is critical to guarantee that VRFs evaluated on the output *seed* cannot be biased by manipulating the seed generation. Intuitively, if the adversary still can bias its own VRF's output, it must can query the VRF oracle (which performs as random oracle [26]) with the right *seed* in a number of polynomial queries (before *seed* is committed), which would raise contradiction to break the unpredictability property.

Remark on totality. The totality property ensures that no honest party would receive some output *seed* solely. I.e., if an honest party gets *seed*, we can assert that all honest parties will do so.

Lemma 8. *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 $\mathcal{O}(n^2)$ messages and $\mathcal{O}(\lambda n^2)$ bits, assuming EUF-CMA secure digital signature and SXDH assumption.*

The recent elegant result of aggregatable public verifiable secret sharing (PVSS) due to Gurkan et al. [40] lift a PVSS scheme to enjoy aggregability. Employing this aggregatable PVSS, we can construct an exemplary Seeding protocol. 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 proof, each party would decrypt its corresponding share from the committed PVSS script, thus ensuring the unpredictability and commitment properties. We defer the proof for Lemma 8 along with the exemplary construction (cf. Alg. 7) in Appendix B.

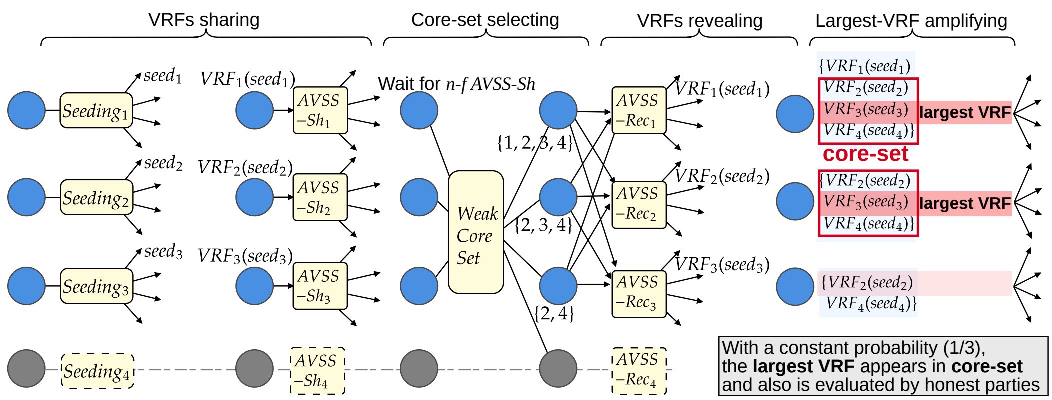

Fig. 2: Overview of our Coin protocol. VRFs (patched by Seeding) are shared via AVSSes. Then, $f+1$ honest parties get a core-set representing $n-f$ completed AVSSes. After that, the VRFs are revealed, and each party multicasts the largest VRF seen by it.

Constructing private-setup free Coin. With Seeding at hand, we are ready to present our Coin construction (formally shown in Alg. 4), which has four main steps:

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. *Core-set selecting* (Line 9-12). 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 activate WCS taking S as input.
3. *VRFs revealing* (Line 13-24). Once WCS outputs $\hat{S}$, an honest party $\mathcal{P}_i$ starts to reconstruct AVSS associated to the indices in $\hat{S}$. After reconstructions, it might get some valid VRF evaluation-proof, and then multicasts the VRF with maximum evaluation to all parties.
4. *Largest-VRF amplifying* (Line 25-31). After receiving $n-f$ Candidate messages encapsulating valid VRF evaluation-proof pairs, $\mathcal{P}_i$ selects the index $\hat{l}$ of the largest evaluation and outputs the lowest bit of $r_{\hat{l}}$.

Security analysis of Coin. The main security properties of Coin can be bridged to the next two key lemmas. Intuitively, Lemma 9 states that every party can choose a speculative largest VRF evaluation. This is because at least $n-f$ Seedings and $n-f$ AVSSes must complete to ensure the completeness of WCS, and implies the termination of Coin. While, Lemma 10 bounds the probability of good case,

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

$S \leftarrow \emptyset$, $\Sigma \leftarrow \emptyset$, $R \leftarrow \emptyset$, $C \leftarrow \emptyset$, $X \leftarrow 0$
$\widehat{S} \leftarrow \perp$, $seed_j \leftarrow \perp$ for each j in $[n]$
activate Seeding[⟨ID, j⟩] for each $j \in [n]$ with being the leader in Seeding[⟨ID, i⟩]
▷ If a trusted nonce η is provided by "genesis" (generated by an initialization functionality after all have registered at PKI, cf. [26]), this line can be replaced by "$seed_j \leftarrow \eta$, for each $j \in [n]$"

upon $seed_j \leftarrow$ Seeding[⟨ID, j⟩] **do**
  if $j = i$ **then**
    $(r, \pi) \leftarrow \mathsf{VRF.Eval}_i^{\mathsf{ID}}(seed_j)$, and activate AVSS-Sh[⟨ID, j⟩] as dealer with taking (r, π) as input
  else
    activate AVSS-Sh[⟨ID, j⟩] as a non-dealer participant

upon receiving output from AVSS-Sh[⟨ID, j⟩] **do**
  $S \leftarrow S \cup \{j\}$
  if $|S| = n - f$ **then**
    activate WCS taking S as input

wait for WCS outputs $\widehat{S}$
  send RecRequest(ID, k) to all parties for every $k \in \widehat{S}$
  wait for that for every $k \in \widehat{S}$, AVSS-Rec[⟨ID, k⟩] 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 Candidate(ID, ℓ, r_ℓ, π_ℓ) to all parties

upon receiving RecRequest(ID, k) from any party for the first time **do**
  wait for $\widehat{S} \neq \perp$ and that AVSS-Sh[⟨ID, k⟩] outputs ss_k **do** ▷ If $\widehat{S}$ becomes $\emptyset$, it is no longer $\perp$
    activate AVSS-Rec[⟨ID, k⟩] with taking ss_k as input

upon receiving Candidate(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 $r_{\hat{\ell}}$

and states that with a constant probability α, all honest parties could decide the same speculative largest VRF evaluation $r_{\hat{l}}$ that is also evaluated by some honest party. This implies that with constant probability, the output bit is as uniformly flipped.

Lemma 9. (Termination) *If all honest parties are activated on* ID*, every honest party will decide a speculative largest VRF evaluation $r_{\hat{l}}$ with a valid proof $\pi_{\hat{l}}$, and all honest parties can eventually receive the same $seed_{\hat{l}}$ that $r_{\hat{l}}$ is evaluated on.*

Proof. According to the correctness and commitment of Seeding, if all honest parties participate in Seeding[⟨ID, j⟩], every party will get the same $seed_j$ regarding an honest $\mathcal{P}_j$, which means that all honest parties can complete their Seedings, compute their VRFs and activate their AVSS-Sh instances that would finally joined by all honest parties. So every honest party will eventually complete at least $n - f$ AVSS-Sh instances and record a $(n - f)$-sized set S including the indexes of which AVSS-Sh instances it participants in. Then every honest party activates WCS taking its set S.

From the totality of AVSS, if some honest party completes AVSS-Sh instance on ID, all honest parties will complete. So any index in some honest party's input set can eventually appear in all honest parties' set. From the termination of WCS, all honest parties will output a set $\hat{S}$ and send RecRequest messages.

From the validity of WCS and the totality of AVSS, all honest parties would complete all AVSS-Sh instances corresponds to its $\hat{S}$ and then start AVSS-Recs. According to the totality and commitment of AVSS, all secrets corresponds to its $\hat{S}$ can be reconstructed. Recall that for each $k \in \hat{S}_i$, $\mathcal{P}_i$ participants in AVSS-Sh$[ID, k]$. An honest party activates an AVSS-Sh$[\mathsf{ID}, k]$ only if it receives a $seed_k$ from Seeding$[\langle \mathsf{ID}, k \rangle]$. This means that for each $k \in \hat{S}$, $\mathcal{P}_i$ can output in 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 a validated VRF result, and pick 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. According to the totality and commitment of Seeding, if any honest party gets $seed_j$ from Seeding$[\langle \mathsf{ID}, j \rangle]$, all honest parties would obtain the same $seed_j$ so all honest parties can get common VRF seeds and mutually consider whether others' Candidate messages are valid, and then output (j, r, π) where r is the maximum of all r_l among valid Candidate messages.

Lemma 10. (Good-case bound) *Let* $\mathsf{Event_{good}}$ *to denote the case in which the (f+1)-supporting core-set* S^* *solicits an honest party's VRF evaluation* r *that is also largest among all parties' VRF. The remaining case denoted by* $\mathsf{Event_{bad}}$ *to cover all other possible executions. Then,* $\Pr[\mathsf{Event_{good}}] \geq \alpha = 1/3$*, under the ideal functionality of VRF in [26] (which is realizable in the random oracle model with CDH assumption).*

Proof. From the $(f+1)$-support core set of WCS, at the moment when the first honest party outputs from WCS and invokes any AVSS-Rec instance, there has existed a core set S^* including $2n/3$ indices, each of which represents a shared VRF's evaluation and at most $f < n/3$ indices of which are shared by adversary. Each VRF's evaluation has a $1/n$ probability to be the maximal. Otherwise, the adversary directly breaks the pseudorandomness of VRF by 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). Thus, the probability that the $\mathsf{Event_{good}}$ occurs is at least $\frac{\frac{2n}{3} - \frac{n}{3}}{n} = \frac{1}{3}$.

Lemma 11. *If* $\mathsf{Event_{good}}$ *defined in Lemma 10 occurs, there does not exist a polynomial adversary which can predicate the lowest bits(e.g. lowest* $\lambda/2$ *bits) of* $r_{\hat{l}}$*.*

Proof. Recall that the honest dealers' AVSS-Recs leak nothing about their VRF's evaluations so at the moment when S^* is fixed, the adversary learns nothing about honest parties' VRF evaluations. Thus when $\mathsf{Event_{good}}$ defined in Lemma 10 occurs, the adversary cannot predicate the lowest bits of output better than guessing. Otherwise it can break the pseudorandomness of VRF by predicating the lowest bits of honest parties VRF's evaluations without accessing their secret keys.

Lemma 12. *If* $\mathsf{Event_{good}}$ *defined in Lemma 10 occurs, all honest parties will output the same bit* b*.*

Proof. When $\mathsf{Event_{good}}$ defined in Lemma 10 occurs, at least $f+1$ honest parties will receive the largest VRF's evaluation r from some honest party and multicast it with Candidate messages. All honest parties can receive at least one Candidate message containing r so that all honest parties will output the lowest bit of r.

Theorem 3. *In the the bulletin PKI setting and the random oracle model, our* Coin *protocol (formally described in Alg. 4) 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.*

Proof. We prove that Alg. 4 realizes the properties of Coin in Def. 3 one by one:

- *Termination.* Termination can be proved directly from Lemma 9,
- *Reasonably fair bit-tossing.* From Lemma 10, the $\mathsf{Event_{good}}$ occurs with a probability $\alpha = 1/3$. 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 Lemma 11, before $f + 1$ honest parties run the protocol, and when $\mathsf{Event}_{\mathsf{good}}$ occurs, the adversary cannot predicate the lowest bit of the largest VRF's evaluation better than guessing. Moreover, in this case, all honest parties output the same bit according to Lemma 12. Therefore, the adversary succeeds in predicating some honest party's output with $\alpha/2$ probability. When $\mathsf{Event}_{\mathsf{bad}}$ occurs with $1 - \alpha$ probability, honest parties may not be able to output the same value, i.e., some honest parties output 0 and some may output 1. In this case, the adversary can always guess a bit b which is equal to some honest parties' output. Therefore, the probability that adversary wins in the predication game is $\Pr[\mathcal{A} \text{ wins}] \leq 1 - \alpha + \alpha/2 = 1 - \alpha/2$, where $\alpha = 1/3$.

Remark on static security. The reason why our common coin cannot tolerate adaptive corruptions is that the Seeding protocol is static, which is further caused by using Gurkan et al.'s statically-secure non-interactive PVSS instantiation [40]. Note that if there is a one-time common random string η announced after PKI registration, we do not have to use Seeding to pack VRF, and it is fine to directly use η as VRF seed (cf. Line 3 in Algorithm). In such setting, our protocol actually can be adaptive secure, because AVSS and WCS are adaptively secure to ensure that: there must exist $f + 1$ *forever* honest parties that hold a set of VRF evaluation-proof pairs with an intersection consisting of at least $n - f$ common VRFs, so Lemmas 9 and 10 still hold against an adaptive adversary in such setting. Namely, given the extra setup assumption of one-time common random string, our coin protocol (and also our later results including Election, ABA and VBA constructions) can be adaptively secure.

Complexities of Coin. The Coin protocol incurs $\mathcal{O}(n^3)$ messages, because it activates one WCS, n AVSSes and n Seedings in addition to n^2 Candidate messages and n^3 RecRequest messages. The overall communication complexity is $\mathcal{O}(\lambda n^3)$, because each AVSS and Seeding incurs $\mathcal{O}(\lambda n^2)$ bits, the WCS protocol exchanges $\mathcal{O}(\lambda n^3)$ bits, and the size of each Candidate and RecRequest messages is $\mathcal{O}(\lambda n)$-bit and $\mathcal{O}(n)$-bit, respectively. Also, the Coin can terminate in constant asynchronous rounds, mainly because all underlying building blocks would output in constant rounds deterministically.

6.2 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 [17,55].

Definition 5 (Asynchronous Binary Agreement). *A protocol realizes* ABA*, if it has syntax and properties defined as follows.*

Syntax. *For all execution of 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 are activated 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 *for* ID*.*

Constructing ABA. We refrain from reintroducing the ABA protocols presented in [55] and [23], as we only need to plug in our Coin primitive to instantiate their reasonably fair common coin abstraction. More formally,

Theorem 4. *Given our* $(n, f, 2f + 1, 1/3)$-Coin *protocol, [55] and [23] can implement* ABA *in the asynchronous message-passing model with* $n/3$ *static Byzantine corruption and bulletin PKI assumption, and cost expected constant asynchronous rounds, 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 [55] and [23], respectively. The complexity of resulting ABAs is dominated by our $(n, f, 2f + 1, 1/3)$-Coin protocol, because given costless Coin, both protocols can terminate in expected constant rounds and exchange only $\mathcal{O}(n^2)$ bits.

7 Augment: Towards Leader Election with Agreement

This Section presents our efficient asynchronous leader election (Election) protocol without relying on private setup. This 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 efficient asynchronous DKG [4], our technique essentially can be plugged in the existing fast-terminating AJM+21 ADKG protocol to reduce its communication complexity from $\mathcal{O}(\lambda n^3 \log n)$ to $\mathcal{O}(\lambda n^3)$, thus initialing a new path to easy-to-deploy replicated services in the asynchronous network.

7.1 Leader election without private setup

The aim of the Election primitive, in our context, is to randomly elect someone of the participating parties [5]. More importantly, the primitive shall prevent the adversary from fully predicating which party would be elected, otherwise, the adversary might schedule message deliveries and cause the higher level protocol to never stop (or at least cause slow termination). For example, in Abraham et al's VBA [5], Election is invoked after $n - f$ input broadcasts are completed, and the termination of this VBA protocol requires Election to luckily choose an indeed completed input broadcast. Clearly, if the adversary can always predicate the Election result in advance, it can always delay the to-be-elected broadcast to make it not appear in the $n-f$ completed broadcasts, thus causing VBA not to terminate.

Necessity of Agreement. In addition, different from Coin that only ensures agreement with a constant probability (e.g., 1/3), Election always has agreement. This is particularly important in many VBA constructions, because Election is usually used to decide which party's input becomes the final output, so lacking agreement in Election might immediately break VBA's agreement. Essentially, the main task of this Section is to lift our Coin protocol to realize the necessary agreement.

Formally, we would design an Election protocol realizing the following properties in the asynchronous network without private setups:

Definition 6 ($(n, f, f + k, \alpha)$-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 next 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 $\ell_i = \ell_j$.*
- **Reasonably fair leader-election**. *Before k honest parties ($1 \leq k \leq f + 1$) are activated on* ID*, the adversary $\mathcal{A}$ cannot always predicate the elected leader. More precisely, consider the predication game: $\mathcal{A}$ guesses 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 require $\Pr[\mathcal{A} \text{ wins}] \leq 1 - \alpha + \alpha/n$.*
  Here α represents the lower-bound probability that the output is as if uniformly distributed over $[n]$, while $1 - \alpha$ captures the possibility that the adversary might predicate/bias the output.

Remarks. When plugging in a $(n, f, f+k, \alpha)$-Election with agreement ($k \geq 1$), most VBA constructions [16,5,52] can preserve their securities and still terminate in expected constant rounds, as long as α is a certain constant between $(0, 1]$. Also, sometimes it can be important to realize larger k (e.g., $f + 1$) to clip the power of asynchronous adversary as in the VBA constructions of [5,52], and we present a $(n, f, 2f + 1, 1/3)$-Election protocol.[5]

[5] Note that $(n, f, 2f + 1, 1/3)$-Election can prevent an adaptive adversary from always proposing the output in some VBA constructions [5,52] (which give the honest parties a chance to output their proposals). Similar to AJM+21 [4], our Election protocol has the potential to realize adaptive security if the underlying aggregatable PVSS can be adaptively secure. So we lift $k = f + 1$ to maximize the strength of results.

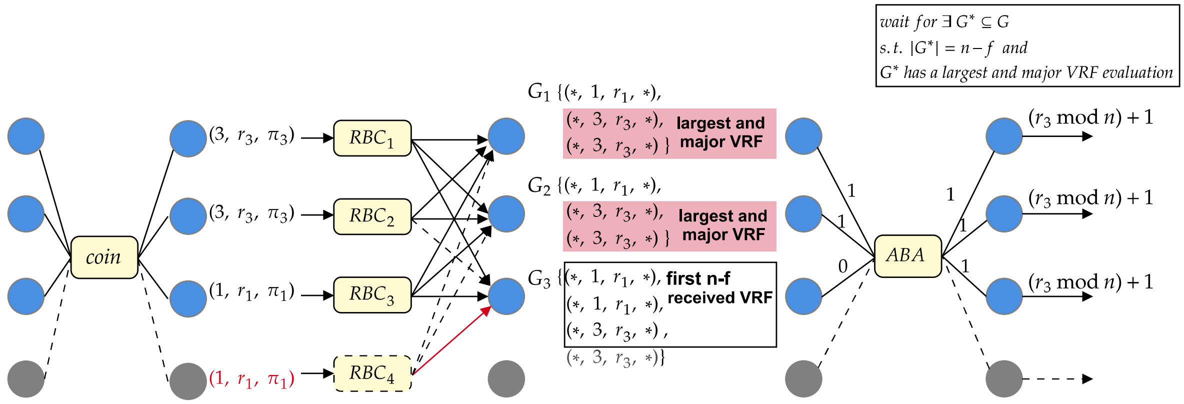

Fig. 3: Overview of our Election protocol. Each party inputs 1 to ABA if there exists a VRF evaluation which is the largest and majority among all RBC outputs. If ABA outputs 1, wait for $G^* \subseteq G_i$ and output.

High-level rationale. Our starting point of constructing Election is our Coin protocol, at the end of which each party can have a speculative largest VRF. Recall Lemma 9 and Lemma 10 that reflect the essential properties of Coin.

Lemma 10 states that: with a constant probability α, the speculative largest VRF of all honest parties is essentially same. So our Coin construction essentially can be thought of a leader election realizing agreement with only α probability. Thus, lifting perfect agreement boils down to the problem of cleaning up the possible disagreement in the else $1-\alpha$ worse cases.

Furthermore, an in-depth thinking on Lemma 9 brings to light that all honest parties can get the seeds needed to verify the speculative largest VRFs of all parties at the end of Coin execution. This hints a possibility that they can cross-check the largest VRFs for each other, and then vote on whether a common largest VRF indeed exists. I.e., each party verifies some speculative largest VRFs from at least $n-f$ parties, and then votes 1 (resp. 0) to an asynchronous binary agreement (ABA), according to whether there exists a largest and majority VRF out of the $n-f$ verified VRFs (resp. or not). This efficient voting procedure through one single ABA can resolve the possible disagreement on the speculative largest VRF at the end of coin execution, because the largest and majority VRF in the $n-f$ verified VRFs must be unique if ABA returns 1.

Algorithm 5 Election protocol with identifier ID, for each party $\mathcal{P}_i$

$G \leftarrow \emptyset$, $G^* \leftarrow \emptyset$, $ballot \leftarrow 0$, activate RBC[⟨ID, j⟩] for each $j \in [n]$
run the code of Coin in Alg. 4 with replacing Line 31 by "$\mathsf{vrf}_{\max} \leftarrow (\hat{\ell}, r_{\hat{\ell}}, \pi_{\hat{\ell}})$"
wait for $\mathsf{vrf}_{\max}$ is assigned by $(\hat{\ell}, r_{\hat{\ell}}, \pi_{\hat{\ell}})$, i.e., Line 31 of modified Coin algorithm is executed
  input $\mathsf{vrf}_{\max}$ to RBC[⟨ID, i⟩]

upon RBC[⟨ID, j⟩] outputs $\mathsf{vrf}_{\max,j} = (\ell_j, r_{\ell_j}, \pi_{\ell_j})$ **do**
  if $\mathsf{VRF.Verify}^{\mathsf{ID}}_{\ell_j}(seed_{\ell_j}, r_{\ell_j}, \pi_{\ell_j}) = 1$ **then** ▷ Verifying VRF implicitly waits for $seed_{\ell_j} \neq \perp$
    $G \leftarrow G \cup (j, \ell_j, r_{\ell_j}, \pi_{\ell_j})$
    if $|G| = n - f$ **then**
      if exist $(\cdot, \ell^*, r^*, \cdot)$ matching the majority elements in G **then**
        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
          $ballot \leftarrow 1$
      activate ABA[ID] with $ballot$ as input
      wait for that ABA[ID] outputs b
        if $b = 1$ **then**
          wait for $\exists\ G^* \subset G$ s.t. $|G^*| = n - f$ and G^* has a largest and major VRF evaluation
            output $(r^* \bmod n) + 1$, where r^* is the largest and majority VRF among G^*
        else output the default index, i.e., 1

Constructing Election. Our a reasonably fair random Election protocol is formally shown in Alg. 5, and it has three main steps that proceeds as follows:

1. *Committing the largest VRF* (Line 1-4). Each party firstly runs the code of Coin protocol, and obtains the speculate largest VRF's evaluation seen in its view, i.e., get $\mathsf{vrf}_{\mathsf{max}} = (\hat{\ell}, r_{\hat{\ell}}, \pi_{\hat{\ell}})$, where $\hat{\ell}$ represents that this speculative largest VRF is evaluated by which party, and $r_{\hat{\ell}}$ and $\pi_{\hat{\ell}}$ are the VRF evaluation and proof, respectively. After that, the party broadcasts $\mathsf{vrf}_{\mathsf{max}}$ to commit this speculative largest VRF evaluation. Here Bracha's reliable broadcasts (RBC) [14] are used to prevent the corrupted parties from committing different speculative largest VRF to distinct parties.
2. *Voting on how to output* (Line 5-13). A party can eventually output in $n - f$ RBCs, each of which can return a valid VRF evaluation-proof pair (which can be verified because the needed *seed* can be waited from Seeding protocols). Then, the party checks if there exists a RBC output s.t. (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 activates ABA with input 1, otherwise, it activates ABA with input 0.
3. *Output decision* (Line 14-17). If ABA outputs 1, then each party waits for that there exists a $(n-f)$-sized subset G^* of all valid speculative largest VRFs received from RBCs, s.t. there exists r^* that is the largest and majority VRF evaluation among all VRFs in G^*, then it outputs $(r^* \bmod n)+1$. If ABA outputs 0, all parties would output a default index, e.g., 1.

Security Analysis of Election. The random leader election protocol presented in Algorithm 5 securely realizes $(n, f, 2f + 1, 1/3)$-Election in the asynchronous message-passing model. Here we prove its securities.

Lemma 13. *For any two parties $\mathcal{P}_i$ and $\mathcal{P}_j$, if there exists $(\cdot, \ell, r, \cdot)$ matching the majority elements in G_i^* and r is the largest VRF evaluation among all elements in G_i^*, and there exists $(\cdot, \ell', r', \cdot)$ matching the majority elements in G_j^* and r' is the largest VRF evaluation among all elements in G_j^*, 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_i^* and G_j^*, respectively, which means that the number of their appearance in G_i^* and G_j^* are at least $f + 1$, respectively. Without loss of generality, we assume that $r > r'$. Note that there are $n - f$ elements in G_j^*, so at least one valid $(\cdot, \ell, r, \cdot)$ must be included in G_j^*, because all elements in G_i^* and G_j^* are obtained via reliable broadcast that ensures agreement. Since r' is the largest VRF evaluation among all elements in G_j^*, it also means $r' > r$, which is a contradiction to the assumption. Hence, $(\ell, r) = (\ell', r')$.

Lemma 14. *For any two honest parties output $b = 1$ from* ABA*, they will output the same elected leader.*

Proof. 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 honest party record a G^* where exists $(\cdot, \ell^*, r^*, \cdot)$ matching the majority elements in G^* and r^* is the largest VRF evaluation among all elements in G^*. Since all elements are the outputs of RBCs. From the totality, all honest parties can receive all elements in this G^*. Hence, each honest party can wait for a $G^* \subseteq G$ and then output. According to the Lemma 13, any valid G^* has 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 $(r \bmod n) + 1$.

Lemma 15. *When the $\mathsf{Event}_{\mathsf{good}}$ defined in Lemma 10 occurs, the polynomial-time adversary cannot predicate the elected leader better than guess.*

Proof. From Lemma 12, when the $\mathsf{Event}_{\mathsf{good}}$ occurs, all honest parties will output the same $\mathsf{vrf}_{\mathsf{max}} = (\ell^*, r^*, \pi^*)$ after running the code of Coin. 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, at least $n - 2f \geq f + 1$ 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 activate ABA with 1 as input. According to the validity of ABA, all honest parties will output 1 from ABA, then all honest parties output the same value $(r^* \bmod n) + 1$ according to Lemma 14.

Following Lemma 11, with a probability $\alpha = \Pr[\mathsf{Event_{good}}] = 1/3$, the adversary cannot predict $\ell = (r^* \bmod n) + 1$ better than guessing. Thus in this case, the probability that the adversary $\mathcal{A}$ succeeds in predicting some honest party's output is no more than $\frac{\alpha}{n}$.

Theorem 5. *In the bulletin PKI setting, Alg. 5 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.*

Proof. Here we prove that Alg. 5 satisfies the properties of Election given in Def. 6 one by one:

- *Termination.* From Lemma 9, each honest party will output a $\mathsf{vrf_{max}} = (\ell^*, r^*, \pi^*)$, then each honest 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 Lemma 14, all honest parties will output a same value.
- *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 Lemma 14, all honest parties will output $(r \bmod n) + 1$, where r is the largest VRF's evaluation.
- *Reasonably fair leader-election.* We use the $\mathsf{Event_{good}}$ and $\mathsf{Event_{bad}}$ defined in the Lemma 10 to discuss this property by two cases. Case (i): With probability α, the $\mathsf{Event_{good}}$ occurs, then from Lemma 15, 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 $\frac{\alpha}{n}$. Case (ii): if the $\mathsf{Event_{bad}}$ occurs, the adversary $\mathcal{A}$ might lead up to different honest parties to obtain different $\mathsf{vrf_{max}}$, 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 and the adversary always predicates the output.
  In sum, the probability that adversary wins in the predication game is $\Pr[\mathcal{A} \text{ wins}] \leq 1 - \alpha + \alpha/n$, where $\alpha = 1/3$.

Complexities of Election. The overall message complexity is expected $\mathcal{O}(n^3)$, because the execution of Coin part spends $\mathcal{O}(n^3)$ messages, n RBC incurs $\mathcal{O}(n^3)$ messages, and the ABA instance costs expected $\mathcal{O}(n^3)$ messages. The overall exchanged bits are $\mathcal{O}(\lambda n^3)$ bits on average, because each RBC instance costs $\mathcal{O}(\lambda n^2)$ bits and the ABA instance incurs expected $\mathcal{O}(\lambda n^3)$ bits. Moreover, the Election protocol can terminate in expected $\mathcal{O}(1)$ asynchronous rounds, which is mainly dominated by the underlying ABA instance.

7.2 Resulting VBA without private setup

Given our Election protocol, we are ready to construct the private-setup free validated Byzantine agreement (VBA), which is essentially a special Byzantine agreement with external validity. As aforementioned, it is the core building block in many asynchronous protocols, such as atomic broadcast [39] and fast-terminating asynchronous DKG [4]. More formally, it can be defined as follows.

Definition 7 (Asynchronous Validated Byzantine Agreement (VBA)).

Syntax. *For each* VBA *instance with an identifier* ID *and a polynomial-time computable global predicate* Q_{ID}*, each party inputs a value (besides the implicit inputs including all public keys and its own private key), and outputs a 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 [16,5,52] rely on a pre-configured non-interactive threshold PRF (tPRF) [17] to implement a Leader Election primitive that can uniformly elect a common party out of all parties. Alternatively, our reasonably fair Election protocol is pluggable in all VBA implementations [16,5,52] to replace tPRF, thus removing the possible unpleasant private setup of it. More formally,

Theorem 6. *In the bulletin PKI setting, given our* $(n, f, 2f + 1, 1/3)$-Election *protocol, [16,5,52] implement* VBA *in the asynchronous message-passing model with* $n/3$ *static Byzantine corruption, and cost expected constant rounds, expected* $\mathcal{O}(n^3)$ *messages, and expected* $\mathcal{O}(\lambda n^3)$ *bits (w.r.t.* λn*-bit or shorter inputs).*

Proofs for VBA properties can be found in [16,5,52] with some trivial adaptions. Here we briefly discuss the security intuition behind the securities and the resulting complexities, with using Abraham et al.'s VBA construction [5] as an example.

Recall that each iteration of Abraham et al.'s VBA [5] 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).

7.3 Applications

Application to asynchronous DKG. The resulting VBA protocols can be plugged in AJM+21 ADKG [4] to reduce the communication to $\mathcal{O}(\lambda n^3)$ bits, with preserving fast termination in expected $\mathcal{O}(1)$ rounds and optimal $n/3$ resilience.

The basic idea (cf. Section 7.5 in [4]) lets each party multicast an aggregatable PVSS hiding a random secret. Then, everyone gathers and combines $n - f$ PVSS from distinct parties. So they can input the aggregated PVSS to one VBA instance (with external validity specified to check the input is indeed PVSS aggregated by $n - f$ parties' contributions). Finally, each party can get a consistent PVSS script returned by VBA, and therefore can decrypt it to get its key share. The resulting communication cost is $\mathcal{O}(\lambda n^3)$, because all PVSS scripts are $\mathcal{O}(\lambda n)$-bit.

Application to random beacon w/o DKG. Our Election protocol can be slightly adapted to realize an asynchronous random beacon service that all participating parties can proceed by consecutive epochs and continually output an unbiased and unpredictable value in each epoch. Here unbiased means the output is uniformly distributed [12,24]; and the unpredictable means that the adversary cannot tell the random output of next epoch better than guessing, unless $f+1$ honest parties already output in the current epoch.

To implement asynchronous random beacon, we can let all parties to execute a sequence of Election protocols, with the following minor changes: (i) when ABA unluckily returns 0 and thus no largest VRF is agreed, the honest parties do not output the default value, and they directly move into the next Election instance; (ii) instead of returning a short index belong $[n]$, the parties can output the lowest $\mathcal{O}(\lambda)$ bits of the selected largest VRF, e.g., the half least-significant bits of the largest VRF evaluation. The former adaption ensures that a non-default value will be output with a probability of $1-(1-\alpha)^k$ after sequentially running k Election instances, and thus a random value can always be output after expected constant rounds.

For bias-resistance, according to the committing and unpredictability of Seeding, the adversary cannot manipulate the generation of VRF seeds so that they cannot bias the VRFs evaluated on the *seed* or immediately break the unpredictability of VRF. The unpredictability is similar, because before $f+1$ honest parties invoke Seeding protocols, the adversary cannot predicate the output VRF seeds, so all VRF evaluations in the next epoch would remain secret to the adversary.

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., Chan, T.H., Dolev, D., Nayak, K., Pass, R., Ren, L., Shi, E.: Communication complexity of byzantine agreement, revisited. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 317–326 (2019)
3. Abraham, I., Devadas, S., Dolev, D., Nayak, K., Ren, L.: Synchronous byzantine agreement with expected $o(1)$ rounds, expected $o(n^2)$ communication, and optimal resilience. In: International Conference on Financial Cryptography and Data Security. pp. 320–334. Springer (2019)
4. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Reaching consensus for asynchronous distributed key generation. In: Proceedings of the 40th Symposium on Principles of Distributed Computing (2021)
5. 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)
6. AlHaddad, N., Varia, M., Zhang, H.: High-threshold avss with optimal communication complexity. In: International Conference on Financial Cryptography and Data Security (2021)
7. Attiya, H., Welch, J.: Distributed computing: fundamentals, simulations, and advanced topics, vol. 19. John Wiley & Sons (2004)
8. Backes, M., Datta, A., Kate, A.: Asynchronous computational vss with reduced communication complexity. In: Topics in Cryptology – CT-RSA 2013. pp. 259–276
9. Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: Advances in Cryptology – ASIACRYPT 2011. pp. 590–609
10. 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)
11. Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: Proceedings of the second annual ACM symposium on Principles of distributed computing. pp. 27–30. ACM (1983)
12. Bhat, A., Shrestha, N., Luo, Z., Kate, A., Nayak, K.: Randpiper – reconfiguration-friendly random beacons with quadratic communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 3502–3524 (2021)
13. 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)
14. Bracha, G.: Asynchronous byzantine agreement protocols. Information and Computation **75**(2), 130–143 (1987)

15. 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)
16. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)
17. 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)
18. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05). pp. 191–201. IEEE (2005)
19. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: Annual International Cryptology Conference. pp. 98–116 (1999)
20. 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)
21. Cascudo, I., David, B.: SCRAPE: Scalable randomness attested by public entities. In: Proc. ACNS 2017. pp. 537–556
22. 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)
23. 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)
24. Das, S., Krishnan, V., Isaac, I.M., Ren, L.: Spurt: Scalable distributed randomness beacon with transparent setup. In: 2022 IEEE Symposium on Security and Privacy (SP)
25. Das, S., Xiang, Z., Ren, L.: Asynchronous data dissemination and its applications. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (2021)
26. 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)
27. Dinsdale-Young, T., Magri, B., Matt, C., Nielsen, J.B., Tschudi, D.: Afgjort: A partially synchronous finality layer for blockchains. In: International Conference on Security and Cryptography for Networks. pp. 24–44. Springer (2020)
28. Dolev, D., Reischuk, R.: Bounds on information exchange for byzantine agreement. Journal of the ACM (JACM) **32**(1), 191–204 (1985)
29. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM Journal on Computing **12**(4), 656–666 (1983)
30. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. J. ACM **35**, 288–323 (1988)
31. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous byzantine agreement. SIAM Journal on Computing **26**(4), 873–933 (1997)
32. Fischer, M., Lynch, N., Merritt, M.: Easy impossibility proofs for distributed consensus problems. Distributed Computing **1**, 26–39 (2005)
33. 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)
34. Fitzi, M., Garay, J.: Efficient player-optimal protocols for strong and differential consensus. In: PODC '03 (2003)
35. Gągol, A., Leśniak, D., Straszak, D., Świętek, M.: Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In: Proceedings of the 1st ACM Conference on Advances in Financial Technologies. pp. 214–228 (2019)
36. Ganesh, C., Patra, A.: Optimal extension protocols for byzantine broadcast and agreement. Distributed Computing **34**(1), 59–77 (2021)
37. Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A., Xiang, Z.: Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In: Proc. FC 2022
38. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding dumbo: Pushing asynchronous bft closer to practice. In: Proc. NDSS 2022
39. 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)
40. Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Aggregatable distributed key generation. In: Advances in Cryptology – EUROCRYPT 2021
41. Kapron, B.M., Kempe, D., King, V., Saia, J., Sanwalani, V.: Fast asynchronous byzantine agreement and leader election with full information. ACM Transactions on Algorithms (TALG) **6**(4), 1–28 (2010)

42. 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)
43. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: International conference on the theory and application of cryptology and information security. pp. 177–194. Springer (2010)
44. Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement pp. 445–462 (2006)
45. Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. In: Proceedings of the 40th Symposium on Principles of Distributed Computing (2021)
46. King, V., Saia, J.: Byzantine agreement in polynomial expected time. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 401–410 (2013)
47. 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)
48. Krawczyk, H.: Secret sharing made short. In: Advances in Cryptology – CRYPTO 1993. pp. 136–146
49. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4**(3), 382–401 (1982)
50. Libert, B., Joye, M., Yung, M.: Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. Theoretical Computer Science **645**, 1–24 (2016)
51. Lu, D., Yurek, T., Kulshreshtha, S., Govind, R., Kate, A., Miller, A.: Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 887–903 (2019)
52. 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)
53. Micali, S., Vadhan, S., Rabin, M.: Verifiable random functions. In: Proceedings of the 40th Annual Symposium on Foundations of Computer Science. p. 120 (1999)
54. 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)
55. 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)
56. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
57. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: 34th International Symposium on Distributed Computing (DISC 2020). Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
58. Patra, A., Rangan, C.P.: Communication optimal multi-valued asynchronous byzantine agreement with optimal resilience. In: International Conference on Information Theoretic Security. pp. 206–226 (2011)
59. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140 (1991)
60. Rabin, M.O.: Randomized byzantine generals. In: 24th Annual Symposium on Foundations of Computer Science (sfcs 1983). pp. 403–409. IEEE (1983)
61. Yang, L., Park, S.J., Alizadeh, M., Kannan, S., Tse, D.: DispersedLedger: High-Throughput byzantine consensus on variable bandwidth networks. In: 19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22) (2022)
62. Yurek, T., Luo, L., Fairoze, J., Kate, A., Miller, A.K.: hbacss: How to robustly share many secrets. IACR Cryptol. ePrint Arch. **2021**, 159 (2021)

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 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 [40]. The general idea of [40] is to lift the beautiful Scrape PVSS scheme [21] 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. 7 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. [40]. Note that we mainly focus on abstracting the needed properties to construct and prove our Seeding protocol.

B.1 Aggregatable PVSS

Gurkan et al. [40] constructed aggregable PVSS by lifting Scrape PVSS scheme from Cascudo et al. [21]. We slightly rephrase Gurkan et al.'s algorithm description, and present it in Alg. 6. Note that the participating parties use the same CRS consisting of (1) a bilinear group description bp (which fixes $g_1 \in \mathbb{G}_1$ and $\hat{h}_1 \in \mathbb{G}_2$), (2) a group element $\hat{u}_1 \in \mathbb{G}_2$ (3) encryption keys $ek_i \in \mathbb{G}_2$ for every party $\mathcal{P}_i$ with corresponding decryption keys $dk_i \in F$ known only to $\mathcal{P}_i$ such that $ek_i = h^{dk_i}$ and (4) verification keys $vk_i \in \mathbb{G}_2$ for every party $\mathcal{P}_i$ with corresponding secret keys $sk_i \in F$ known only to $\mathcal{P}_i$ such that $vk_i = {g_1}^{sk_i}$.

Conditioned on SXDH assumption and PKI setup, the above aggregatable PVSS scheme has a few nice security properties such as *verifiable commitment*, *verifiable aggregation* and *secrecy*:

- **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 $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{vk}, \mathsf{pvss}) = 1$, 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 VrfyShare with regard to the pvss script from distinct parties (including up to f malicious ones) can recover a secret $F(0)$ same to $F^*(0)$.

Algorithm 6 Implementation of Aggregatable PVSS [40]

$\mathsf{Deal}(\mathsf{ek}, sk_i, a_0) \to \mathsf{pvss}$
- **initialize** $(w_1, .. , w_n) \leftarrow (0, ..., 0)$; $(C_1, .. , C_n) \leftarrow (\perp, ..., \perp)$; $(\sigma_1, .. , \sigma_n) \leftarrow (\perp, ..., \perp)$,
- $w_i \leftarrow 1$; $C_i \leftarrow {g_1}^{a_0}$; $\sigma_i \leftarrow \mathsf{SoK.Sign}(C_i, sk_i, c_i)$
- randomly choose $(a_1, ..., a_t)$ from $\mathbb{F}^t$
- $F(X) \leftarrow \sum_{i=0}^{t} a_i X^i$
- $F_0, ..., F_t \leftarrow {g_1}^{a_0}, ..., {g_1}^{a_t}$; $\hat{u}_2 \leftarrow \hat{u}_1^{a_0}$; $A_1, ..., A_n \leftarrow {g_1}^{f(\omega_1)}, ..., {g_1}^{f(\omega_n)}$; $\hat{Y}_1, ..., \hat{Y}_n \leftarrow {ek_1}^{f(\omega_1)}, ..., {ek_n}^{f(\omega_n)}$
- **return** $\mathbf{F}$, $\hat{u}_2$, $\mathbf{A}$, $\hat{\mathbf{Y}}$, $(C_1, ..., C_n)$, $(w_1, ..., w_n)$, $(\sigma_1, ..., \sigma_n)$

$\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{vk}, \mathsf{pvss}) \to 0/1$
- $\mathbf{F}$, $\hat{u}_2$, $\mathbf{A}$, $\hat{\mathbf{Y}}$, $\mathbf{C}$ $\boldsymbol{w}$, $\boldsymbol{\sigma}$ $\leftarrow$ **parse**(pvss)
- $\alpha \xleftarrow{\$} \mathbb{F}$
- **check** $\prod_{j=1}^{n} {A_j}^{l_j(\alpha)} = \prod_{j=0}^{t} F_j^{\alpha^j}$, $l_j(X)$ denotes the Lagrange polynomial equal to 1 at ω_j and 0 at $\omega_i \neq \omega_j$
- **check** $e(F_0, \hat{u}_1) = e(g_1, \hat{u}_2)$
- **check** $e(g_1, \hat{Y}_j) = e(A_j, ek_j)$ for each $1 \leq j \leq n$
- **for** $1 \leq i \leq n$: **if** $w_i \neq 0$, then **check** $\mathsf{SoK.Vrfy}(vk_i, C_i, \sigma_i) = 1$
- **check** $C_1^{w_1} ... C_n^{w_n} = F_0$
- **return** 1 if all checks pass, else **return** 0

$\mathsf{AggScripts}(\mathsf{pvss}_1, \mathsf{pvss}_2) \to \mathsf{pvss}$
- $(F_{1,0}, ..., F_{1,t})$, $\hat{u}_{1,2}$, $(A_{1,1}, ..., A_{1,n})$, $(\hat{Y}_{1,1}, ..., \hat{Y}_{1,n})$, $(C_{1,1}, ..., C_{1,n})$, $(w_{1,1}, ..., w_{1,n})$, $(\sigma_{1,1}, ..., \sigma_{1,n}) \leftarrow$ **parse**(pvss_1)
- $(F_{2,0}, ..., F_{2,t})$, $\hat{u}_{2,2}$, $(A_{2,1}, ..., A_{2,n})$, $(\hat{Y}_{2,1}, ..., \hat{Y}_{2,n})$, $(C_{2,1}, ..., C_{2,n})$, $(w_{2,1}, ..., w_{2,n})$, $(\sigma_{2,1}, ..., \sigma_{2,n}) \leftarrow$ **parse**(pvss_2)
- **for** $0 \leq i \leq t$: $F_i \leftarrow F_{1,i} F_{2,i}$
- **for** $1 \leq i \leq n$:
  - $A_i \leftarrow A_{1,i} A_{2,i}$; $\hat{Y}_i \leftarrow \hat{Y}_{1,i} \hat{Y}_{2,i}$; $w_i \leftarrow w_{1,i} + w_{2,i}$
  - **if** $\sigma_{1,i} \neq \perp$: $\sigma_i \leftarrow \sigma_{1,i}$, **else**: $\sigma_i \leftarrow \sigma_{2,i}$
  - **if** $C_{1,i} \neq \perp$: $C_i \leftarrow \sigma_{1,i}$, **else**: $C_i \leftarrow \sigma_{2,i}$
- $\hat{u}_2 \leftarrow \hat{u}_{1,2} \hat{u}_{2,2}$
- **return** $\mathbf{F}$, $\hat{u}_2$, $\mathbf{A}$, $\hat{\mathbf{Y}}$, $\mathbf{C}$, $\boldsymbol{w}$, $\boldsymbol{\sigma}$

$\mathsf{GetShare}(dk_i, \mathsf{pvss}) \to \mathsf{sh}_i$
- $\mathbf{F}$, $\hat{u}_2$, $\mathbf{A}$, $\hat{\mathbf{Y}}$, $\mathbf{C}$, $\boldsymbol{w}$, $\boldsymbol{\sigma}$ $\leftarrow$ **parse**(pvss)
- **return** $\hat{\mathbf{Y}}_{\mathbf{i}}^{{dk_i}^{-1}}$

$\mathsf{VrfyShare}(j, \mathsf{sh}_j, \mathsf{pvss}) \to 0/1$.
- $\mathbf{F}$, $\hat{u}_2$, $\mathbf{A}$, $\hat{\mathbf{Y}}$, $\mathbf{C}$, $\boldsymbol{w}$, $\boldsymbol{\sigma}$ $\leftarrow$ **parse**(pvss)
- **check** $e(A_j, \hat{h}_1) = e(g_1, \mathsf{sh}_j)$
- **return** 1 if the check pass, else **return** 0

$\mathsf{AggShares}(\{(j, \mathsf{sh}_j)\}_t) \to a$
- $S \leftarrow \emptyset$
- **for** all input (j, sh_j): **if** $\mathsf{VrfyShare}(j, \mathsf{sh}_j, \mathsf{pvss}) = 1$, then $S \leftarrow S \cup (j, \mathsf{sh}_j)$
- **return** $\prod_{i \in S} {\mathsf{sh}_i}^{l_{S,i}(0)}$

$\mathsf{VrfySecret}(s, \mathsf{pvss}) \to 0/1$
- **check** $e(F_0, \hat{h}_1) = e(g_1, s))$
- **return** 1 if the check pass, else **return** 0

$\mathsf{Weights}(\mathsf{pvss}) \to \boldsymbol{w}$
- $\mathbf{F}$, $\hat{u}_2$, $\mathbf{A}$, $\hat{\mathbf{Y}}$, $\mathbf{C}$, $\boldsymbol{w}$, $\boldsymbol{\sigma}$ $\leftarrow$ **parse**(pvss)
- **return** $\boldsymbol{w}$

- **Verifiable aggregation** means that any verified PVSS script pvss with weight tag $\boldsymbol{w}$ must indeed commit a secret that is a linear combination of participating parties' secrets due to $\boldsymbol{w}$.

  - If $\mathcal{P}_i$ is honest, it is infeasible for the adversary to compute a PVSS script pvss_i by itself (without query $\mathcal{P}_i$ to get pvss_i) s.t., $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{vk}, \mathsf{pvss}_i) = 1$ and $\mathsf{Weights}(\mathsf{pvss}_i)$ returns $\boldsymbol{w}$ with a non-zero i^{th} position.

- Moreover, $F^*(0) = \sum_{i=1}^{n} w_i F_i^*(0)$. Here $F^*(0)$ is the secret committed to a PVSS script pvss, w_i is the i^{th} element in $\boldsymbol{w} = \mathsf{Weights}(\mathsf{pvss})$, and $F_i^*(0)$ represents the secret that is committed to some PVSS script pvss_i that is computed by the party $\mathcal{P}_i$.

– **Secrecy** means that the adversary learns nothing besides the public knowledge through the PVSS script pvss, unless the adversary gets the decrypted secret shares of $t - f$ honest parties.

With the above properties at hand, we actually see that the adversary to have negligible probability to win the following Prediction game. The property is also called unpredictability by us, and captures the major threats in our Seeding protocol (soon to be explained). It intuitively states that if each honest party $\mathcal{P}_i$ randomly chooses the input secret s_i committed to its PVSS script pvss_i, then the adversary cannot compute the aggregated secret s committed to any valid pvss, as long as pvss has a non-zero weight to reflect some honest party's PVSS script pvss_i is indeed aggregated to it.

Prediction game. *The Prediction 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 $\boldsymbol{w}$ 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.

Lemma 16 (Unpredictability). *There exists an aggregatable* PVSS *construction [40] based on SXDH assumption, such that no static adversary controlling up to f parties can win Prediction game with all but negligible probability.*

Note that Gurkan et al. [40] did not formalize unpredictability for their aggregatable PVSS scheme in our way. Nevertheless, as a trivial corollary, their construction satisfies the unpredictability property (that can simplify the proof for the security of our Seeding protocols), because failing to satisfy unpredictability would directly break the authors' DKG scheme (as the adversary can directly tell the aggregated secret key to break DKG).

B.2 Seeding from Aggregatable PVSS

Here down below we describe an exemplary Seeding construction from the $(n, 2f + 1)$ aggregatable PVSS scheme (as detailed by Alg. 7). 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 I – Seed aggregation* (Line 1-2, 18-22).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 AggPvss message.

– *Committing Phase II – Seed commitment* (Line 3-8, 23-27). After receiving a AggPvss 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 AggPvssCommit message containing a signature set Σ. After receiving a valid AggPvssCommit messages from the leader, each party confirms that the output is actually fixed, and it takes the pvss as a input in GetShare to output a share sh_i regarding the secret committed to pvss.

Algorithm 7 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, $\mathsf{pvss}_i \leftarrow \mathsf{Deal}(\mathsf{ek}, sk_i, s)$, and **send** $\textsc{PvssScript}(\mathsf{ID}, \mathsf{pvss}_i)$ to $\mathcal{P}_L$

3: **upon** receiving $\textsc{AggPvss}(\mathsf{ID}, \mathsf{pvss})$ message from $\mathcal{P}_L$ for the first time **do**
4: **if** $\mathsf{VrfyScript}(\mathsf{ek}, \mathsf{vk}, \mathsf{pvss}) = 1 \wedge$ ($\mathsf{Weights}(\mathsf{pvss})$ contains $2f+1$ ones) **then**
5: record pvss, $\sigma_i \leftarrow \mathsf{Sign}_i^{\mathsf{ID}}(\mathsf{pvss})$ and **send** $\textsc{AggPvssStored}(\mathsf{ID}, \sigma_i)$ to $\mathcal{P}_L$

6: **upon** receiving $\textsc{AggPvssCommit}(\mathsf{ID}, \Sigma)$ from $\mathcal{P}_L$ for the first time **do**
7: **if** Σ contains $2f+1$ valid signatures for a recorded pvss from distinct parties **then**
8: $\mathsf{sh}_i \leftarrow \mathsf{GetShare}(\mathsf{pvss})$ and **send** $\textsc{SeedShare}(\mathsf{ID}, \mathsf{sh}_i)$ to $\mathcal{P}_L$

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

12: **upon** receiving $2f+1$ $\textsc{SeedEcho}(\mathsf{ID}, \mathit{seed})$ from distinct parties **do**
13: **send** $\textsc{SeedReady}(\mathsf{ID}, \mathit{seed})$ to all parties if $\textsc{SeedReady}$ not sent yet

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

16: **upon** receiving $2f+1$ $\textsc{SeedReady}(\mathsf{ID}, \mathit{seed})$ from distinct parties **do**
17: **output** *seed*

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

18: **upon** receiving $\textsc{PvssScript}(\mathsf{ID}, \mathsf{pvss}_j)$ from $\mathcal{P}_j$ for the first time **do**
19: **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**
20: $K \leftarrow K \cup \{\mathsf{pvss}_j\}$
21: **if** $|K| = 2f+1$ **then**
22: $\mathsf{pvss} \leftarrow \mathsf{AggScripts}(K)$ and **send** $\textsc{AggPvss}(\mathsf{ID}, \mathsf{pvss})$ to all parties

23: **upon** receiving $\textsc{AggPvssStored}(\mathsf{ID}, \sigma_j)$ message from $\mathcal{P}_j$ for the first time **do**
24: **wait** for pvss is recorded
25: **if** $\mathsf{SigVerify}_j^{\mathsf{ID}}(\mathsf{pvss}, \sigma_j) = 1$ **then**
26: $\Sigma \leftarrow \Sigma \cup \{(j, \sigma_j)\}$
27: **if** $|\Sigma| = 2f+1$ **then send** $\textsc{AggPvssCommit}(\mathsf{ID}, \Sigma)$ to all parties

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

– *Revealing Phase – Seed recovery* (Line 9-17, 28-31). 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 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 $\textsc{SeedEcho}$ to all parties. After receiving $2f+1$ $\textsc{SeedEcho}$ messages, honest parties send $\textsc{SeedReady}$ to all parties. If an honest party receives $f+1$ $\textsc{SeedReady}$ messages of same value and $\textsc{SeedReady}$ has not been sent, it sends $\textsc{SeedReady}$. When receiving $2f+1$ $\textsc{SeedReady}$ messages of same value, honest parties output the *seed*.

Lemma 17. *Upon any honest party completes the committing phase, any two valid* $\textsc{AggPvssCommit}$ *messages* $\textsc{AggPvssCommit}(\mathsf{ID}, \Sigma)$ *and* $\textsc{AggPvssCommit}(\mathsf{ID}, \Sigma')$ *must have* Σ *and* Σ' *containing* $n-f$ *valid signatures for the same* pvss*, and there exists a fixed value seed associated to this* pvss*.*

Proof. Assume that $\mathcal{P}_i$ is the first party who starts to run the revealing phase of $\mathsf{Seeding}$ protocol, it implies that $\mathcal{P}_i$ received a valid $\textsc{AggPvssCommit}(\mathsf{ID}, \Sigma)$ messag from leader $\mathcal{P}_L$. If another honest party $\mathcal{P}_j$ also received a valid message $\textsc{AggPvssCommit}(\mathsf{ID}, \Sigma')$ from leader $\mathcal{P}_L$, where signatures in Σ are signed for pvss while signatures in Σ' are signed for pvss', since a valid Σ contains $2f+1$

valid signatures for a same pvss from distinct parties. According to the the unforgeability of digital signatures, it induces that at least one honest party signed for both pvss and pvss′, which is impossible because each honest party signs at most once. Hence, when some honest party $\mathcal{P}_i$ starts to run the revealing phase, the Σ from any valid AGGPVSSCOMMIT(ID, Σ) message is for the same pvss. Following the commitment of the PVSS scheme, there exists a fixed value *seed* corresponding to the pvss.

Lemma 18. *If any twos honest party output for* ID*, then they output the same value.*

Proof. Suppose that some honest party outputs $seed'$ from the 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(ID, Σ, $seed'$) 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, $seed' = seed$.

Deferred proofs for Lemma 8. Here we prove that the protocol in Alg. 7 satisfies all properties of Reliable Broadcasted Seeding given in Definition 4 and analyze how does it incur only quadratic messages and communications.

Proof. Here prove that Alg. 7 satisfy the Seeding properties one by one:

- *Totality.* Assume that an honest party outputs in the 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 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 Σ and *seed* that pass verifications. So they would broadcast the same SEEDECHO message and the same SEEDREADY messages, thus finally outputting in the Seeding instance.
- *Commitment.* From Lemma 17, upon any honest party completes the protocol's committing phase, there exists a fixed value *seed* corresponding to the unique pvss. From Lemma 18, if any honest party outputs for ID, then it outputs value *seed*. Thus the commitment can be proved.
- *Unpredictability.* Prior to $f+1$ honest parties are activated to run the revealing phase of the 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 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.