Circular Dragon: $\widetilde{\text{O}}$ptimal Adaptively Secure Distributed Randomness/Key Generation and Parallel Broadcast

Hanwen Feng[1], Yuan Lu[2], Tiancheng Mai[3], and Qiang Tang[3]

[1] University of Canterbury, Christchurch, New Zealand
hanwen.feng@canterbury.ac.nz
[2] Agency for Science, Technology and Research (A*STAR), Singapore
lu_yuan@a-star.edu.sg
[3] The University of Sydney, Sydney, Australia
tiancheng.mai@sydney.edu.au, qiang.tang@sydney.edu.au

Abstract. We present the first strongly adaptive common coin protocol, under the PKI setup (in the random oracle model and the algebraic group model, yet with no trusted dealer), that (i) achieves the optimal resilience $t < n/2$ in synchronous networks, and (ii) attains near-optimal asymptotics: near-quadratic communication $O(\kappa\lambda n^2 \log n)$ and logarithmic rounds $O(\kappa \log n)$, with failure probability $\mathsf{negl}(\lambda) + 2^{-\kappa}$, where λ (resp. κ) is the computational (resp. statistical) security parameter. Furthermore, the coin protocol assumes no memory erasure. At its core is a "special-purpose" distributed key generation (DKG) protocol that produces a secret in an elliptic-curve group.

Our common coin immediately implies the first strongly adaptive and optimally resilient Byzantine Agreement with subcubic communication and sublinear rounds in the absence of a trusted dealer. Furthermore, we give two non-trivial applications of independent interest: (i) a strongly adaptive Parallel Broadcast (PBC) under standard PKI with near-quadratic communication and logarithmic rounds, whereas all existing subcubic solutions (Gelles et al. CRYPTO'25, Civit et al. PODC'24) require at least linear rounds; (ii) a strongly adaptive "general-purpose" DKG for field-element secrets with $O(\lambda\kappa n^2 \log n)$ communication and $O(\kappa \log n)$ rounds (assuming memory erasures), fully compatible with legacy threshold schemes (BLS, Schnorr, ElGamal), improving upon (Feng et al. PODC'24), which requires $O(\lambda\kappa n^{2.5} \log n)$ communication and $O(n)$ rounds.

Our core is a new design methodology, which we call *Circular Dragon*. This extends the Dragon method (PODC'24) that was originally developed for sub-cubic communication DKG protocols but with linear rounds, by further reducing the round complexity.

1 Introduction

Randomness is foundational to computation. In distributed systems we often require a public randomness (a.k.a, a common coin) [44,69] that (i) is available to all parties, (ii) cannot be predicted in advance, and (iii) cannot be biased even if some nodes are Byzantine. Such common coins are essential for randomized distributed protocols to achieve better efficiency [73], circumvent certain impossibilities [49], enhance network security [2], or promote fairness [11].

This paper studies common coin protocols and their applications in the synchronous network model, where there is a known upper bound on message delivery time. Our goal is to construct *communication-efficient* and *round-efficient* protocols that achieve (i) honest-majority resilience ($t < n/2$ Byzantine corruptions among n parties)[1], (ii) *strongly* adaptive security [6,52][2], and (iii) reliance on standard PKI and transparent parameters, without a trusted dealer. In particular, we focus on the cost of generating *one coin* from scratch,

[1] For the threshold-coin approach used here and in [16], unpredictability requires a reconstruction threshold $q > t$, while availability without corrupted parties requires $q \leq n - t$. Thus $n \geq 2t + 1$ is optimal for this threshold approach, rather than a universal bound for authenticated broadcast.

[2] That is, the adversary can retract messages sent by a newly corrupted node before they are delivered, a so-called "after-the-fact-removal" attack.

rather than the *amortized* cost over multiple coins. Reducing the cost of single-shot coin generation is not only theoretically significant, since it addresses the fundamental expense of producing public randomness independent of batching techniques, but also practically important, as it lowers the bootstrapping cost of many distributed systems and supports scenarios where only a few coins are needed, such as sublinear-round Byzantine Agreement [7, 63].

Despite extensive research and recent progress in both cryptography and distributed computing, a significant gap remains. Most existing common coin protocols [32] require at least $O(n^3)$ communication and $O(n)$ rounds (in the network size n). Reducing communication complexity and reducing round complexity pose different challenges, and current approaches to address them are even conflicting[3]. For example, Shrestha et al. [70] propose a protocol with expected constant rounds but cubic communication, while two recent works [16, 46] achieve *sub-cubic* communication complexity but require linear rounds.

This status leaves open a central question:

Can we design a strongly adaptive common coin protocol tolerating $t < n/2$ corruptions with $\widetilde{O}(n^2)$ communication and $\widetilde{O}(1)$ rounds without trusted setup?

1.1 Our Results

We answer the above question affirmatively by presenting a *near-optimal* common coin construction. We also develop non-trivial applications of our common coin which will be of independent interest.

Main result: near-optimal common coin via multi-step self-bootstrapping. We propose a common-coin protocol with communication complexity $O(\kappa\lambda n^2 \log n)$ and round complexity $O(\kappa \log n)$, where λ is the computational security parameter and κ is the statistical security parameter. Under the co-OMDL assumption [18] and the SXDH assumption, in the Algebraic Group Model (AGM) and the random-oracle model, our protocol is secure against any polynomial-time *strongly adaptive* adversary corrupting up to $\lfloor (n-1)/2 \rfloor$ parties; notably, it does *not* require memory erasures. We compare our protocol with prior synchronous solutions in Table 1.[4]

At the core of our construction is a distributed key generation (DKG) protocol that distributes a secret key in an elliptic-curve group and realizes the setup required by the one-round common coin of [16]. This "special-purpose" DKG can both generate a *single-shot* common coin directly from the secret key and bootstrap [16] for subsequent generation of multiple one-round common coins.

To achieve this, we develop a set of "self-bootstrapping" techniques that first establish *endemic coins*, i.e., common coins available only within small subsets of parties, and then leverage them to construct common coins for progressively larger participant sets. We further prove that the resulting components compose securely. See the technical overview (§ 2) for details.

Applications. Our common coin enables various near-optimal distributed protocols, through immediate or non-trivial applications.

Immediate application: Byzantine Agreement. Byzantine agreement (BA) [64] is a fundamental problem in distributed computing and cryptography, enabling a group of nodes, each with an input value, to agree on a common output despite some nodes being Byzantine. The state-of-the-art synchronous BA protocol is due to Abraham et al. [7], achieving expected $O(n^2)$ communication and $O(1)$ rounds, but relying on a setup for providing common coins. Without such a setup, the communication complexity becomes $O(\lambda n^3)$. By instantiating the common-coin subroutine in [7] with our protocol, we obtain the first strongly adaptive BA protocol tolerating $t < n/2$ corruptions under standard PKI and transparent public parameters, with expected subcubic $O(\kappa\lambda n^2 \log n)$ communication and sublinear $O(\kappa \log n)$ rounds.

[3] We summarize existing synchronous common coin protocols in Table 1, which illustrates the current landscape. Further discussions on existing techniques and recent development are provided in § 2 and § 8.1

[4] In the asynchronous or partial synchronous setting, most protocols also require at least cubic communication, with an exception of very recent works [4, 5, 48], which either require a stronger setup than ours or *memory erasures*, and their techniques do not apply when $n \geq 2t + 1$; see § 8.2 for further discussion.

Table 1. Comparison with synchronous common coin protocols.

Protocol	Setup	Resi.	Comm.Cost	Round	Adap.?
Instantiation-I [33] + [54]	PKI& CRS	1/2	$O(\lambda n^2)$	$O(n)$	✓
Instantiation-II [33] + [13]	PKI	1/2	$O(\lambda n^2 \mathsf{poly}(\log n))$	$O(n)$	✗
OptRand [18, 21]	PKI	1/2	$O(\lambda n^3)$	$O(n)$	✓
BRandPiper [22]	PKI	1/2	$O(\lambda n^4)$	$O(n)$	✓
Drand [1, 17]	Trusted*	1/2	$O(\lambda n^2)$	1	✓
Grandline [16]	Trusted*	1/2	$O(\lambda n^2)$	1	✓
Classical DKGs [56, 68]	PKI	1/2	$O(\lambda n^3 \log n)$ †	$O(n)$	✓
GJM+ [60]	PKI	$\log n/n$	$O(\lambda n^2 \log^2 n)$ †	$O(n)$	✗
SBKN [70]	PKI	1/2	$O(\lambda n^3 \log n)$	$O(1)$	✗
Dragon-DKG-I [46]	PKI	1/2	$O(\lambda n^{2.5} \log n)$	$O(n)$	✓
Grand-DKG [16]	PKI	1/2	$O(\lambda n^2 \log n)$	$O(n)$	✓
Ours	PKI‡	1/2	$O(\kappa\lambda n^2 \log n)$	$O(\kappa \log n)$	✓

`Protocol`: We overview four categories of constructions (divided by lines): (i) Instantiations of the "commit-and-reveal" framework [24, 59]; (ii) The leader-based protocols; (iii) Threshold VRF based solutions; (iv) DKG protocols which imply coins. See § 8.1 for further discussion.
`Setup`: ‡ Our PKI setup includes a URS and transparent public parameters, with no secret trapdoor or trusted dealer (§ 3). * A trusted dealer generates keys for users, which can be possibly eliminated by DKG protocols at the cost of increased complexities.
`Resi.`: the maximal fraction of Byzantine nodes the protocol can tolerate.
`Comm.Cost` measures the number of bits sent by all honest nodes. λ and κ are the computational security parameter and the statistical security parameter, respectively. † These protocols leverage *broadcast* channels in a blackbox manner; In § 8.3, we discuss the instantiation choices.
`Round`: SBKN [70] requires $O(1)$ rounds in *expectation*, while ours can terminate within $O(\kappa \log n)$ rounds with overwhelming probability.

Parallel Broadcast. In many MPC protocols, a key step is that every node *broadcasts* a value, after which all nodes receive the same set of values through the broadcast. Parallel Broadcast (PBC) [10, 55], also known as Interactive Consistency [20, 33, 46], is the primitive to realize this broadcast mechanism.

While recent works [10, 33, 46, 55] have made significant progress, the state-of-the-art strongly adaptive PBC protocols still require either cubic communication [10] or linear rounds [33], even under a trusted setup. Most recently, Gelles et al. [55] proposed a PBC protocol with near-quadratic communication under a standard PKI setup and using only basic tools; however, it requires polynomially many rounds and provides only weak adaptive security (i.e., it does not withstand "after-the-fact removal" attacks); Civit et al.'s quadratic solutions [34] assume static corruption instead.

Building on our common coin and a refined design methodology, we present the first *strongly adaptive* PBC protocol tolerating $t < n/2$ corruptions that achieves expected subcubic communication complexity $O(\kappa n^2 \ell + \kappa\lambda n^2 \log n)$ and sublinear round complexity $O(\kappa \log n)$ under standard PKI and transparent public parameters, where ℓ is the bitlength of an input value. A detailed comparison with prior work appears in Table 2. We present the PBC construction in § 6.

Field-element DKG. While our common coin is based on a *group-element* DKG, most legacy threshold schemes in the DL setting (such as Threshold Schnorr, BLS, and ElGamal) require a field-element secret $sk \in \mathbb{Z}_p$ (with $pk = g^{sk} \in \mathbb{G}$). The most communication-efficient *field-element* DKG to date is due to [46] (i.e., Dragon-DKG-II as shown in Table 3), which achieves $O(\kappa\lambda n^{2.5} \log n)$ communication with $O(n)$ rounds.

Building on our common coin and a refined design methodology,[5] for $2\kappa \leq n$ we present a field-element DKG that achieves expected subcubic communication complexity $O(\kappa\lambda n^2 \log n)$ and sublinear round complexity $O(\kappa \log n)$. This DKG satisfies robustness and the oracle-aided algebraic simulatability notion of [17] against strongly adaptive adversaries, with *memory erasures*. It also satisfies the standard *full secrecy* property [56, 62] in the static setting, while producing the standard field-element key structure required by threshold DL schemes. We present our construction in § 7 and compare it with existing DKGs in Table 3 in § 8.4.

Organization. The rest of the paper is organized as follows. Section 2 gives a technical overview of our approach. Section 3 presents the model, the common coin definition, and the cryptographic and consensus building blocks, including aggregatable PVSS. Section 4 defines and constructs PCSBB, our core consensus component, and proves its security. Section 5 recalls the one-round common coin of [16], presents a warm-up

[5] This construction is similar to the recent asynchronous DKG protocol of [45], while tailored to tolerate $n \geq 2t + 1$ in the synchronous setting.

Table 2. Comparison with synchronous PBC protocols.

Protocol	Setup	Resi.	Comm.Cost	Round	Adap.?
NRS+ [67]	PKI	$1-\epsilon$	$O(n^2\ell + \lambda n^3 + n^4)$	$O(n)$	✓
TLP-I [72]	PKI	$1-\epsilon$	$O(\kappa\lambda n^3 \ell \log n)$	$O(n \log n)$	weak**
TLP-II [72]	Trusted	$1-\epsilon$	$O(\kappa^3 \lambda n^2 \ell \log n)$	$O(\kappa \log n)$	weak**
CDL+25 [36]	Trusted	$1-\epsilon$	$O(\lambda n^2 \ell + (n^3\lambda + n^2\lambda^2)\mathsf{poly}(\log n))$	$O(\kappa \log n)$	weak**
AAPP [3]	None	1/3	$O(n^2\ell + n^4 \log n)$	$O(1)$	✗
ANS-I [10]	PKI	1/2	$O(n^2\ell + \lambda n^3 \log n)$	$O(1)$	✗
ANS-II [10]	Trusted	1/2	$O(n^2\ell + \lambda n^3 \log n)$	$O(1)$	✓
Dragon-IC [46]	PKI	1/2	$O(n^{2.5}\ell + \lambda n^{2.5} \log n)$	$O(n)$	✓
CDG+ [33]	PKI	1/2	$O(n^2\ell + \lambda n^2 \log n)$	$O(n)$	✓
GLL+ [55]	PKI	1/2	$O(n^2\ell \log n + \kappa\lambda n^2 \log^4 n)$	$\mathsf{poly}(n)$	weak**
CEG-I [34]	Trusted	1/2	$O(n^2(\ell+\kappa))$	$O(1)$	✗
CEG-II [34]	PKI	$1-\epsilon$	$O(n^3(\ell+\kappa) + n^4)$	$O(1)$	✗
Our PBC	Trusted	1/2	$O(\kappa n^2 \ell + \kappa\lambda n^2 \log n)$	$O(\log \kappa)$	✓
Our PBC + our coin	PKI‡	1/2	$O(\kappa n^2 \ell + \kappa\lambda n^2 \log n)$	$O(\kappa \log n)$	✓

‡ Our PKI setup includes a URS and transparent public parameters (§ 3), with no secret trapdoor or trusted dealer. For both of our rows, `Comm.Cost` is expected. The round bound with trusted setup is expected; with our coin, it holds with overwhelming probability when $\log N = O(\kappa)$ for the total invocation bound in Lemma 2.
`Adap.?`: ** The adversary cannot perform the "after-fact-removal" attacks.

group-element DKG with sub-linear rounds, and then gives our recursive DKG RDKG, which yields our near-optimal common coin, together with its full security proof. Sections 6 and 7 present our two applications: the strongly adaptive PBC protocol and the field-element DKG, respectively. Section 8 discusses related work, and Section 9 concludes with future directions.

2 Technical Overview

Three stages to the main common-coin result. We explain our construction in three stages, each addressing one bottleneck of the preceding design.

1. *From cubic to subcubic communication.* We first recall the Dragon methodology [46]. Dragon reduces the cubic communication of conventional DKG to subcubic communication. Its deterministic grouping, local aggregation, and consortium-sender Byzantine broadcast (CSBB) are fundamental design ideas on which our construction builds.
2. *From linear to sublinear rounds.* Dragon still has linear round complexity because CSBB uses deterministic consensus. Existing sublinear-round randomized consensus protocols, by contrast, rely on variants of a *global common coin*; constructing such a coin efficiently is precisely our goal. We break this circularity with parallel consortium-sender Byzantine broadcast (PCSBB). PCSBB jointly realizes the functionality of the CSBB instances while preserving subcubic communication and requiring only sublinear rounds. It relies only on *endemic coins*—common coins available within individual consortia—which are cheaper to generate because each consortium is smaller than the full network. Substituting PCSBB for CSBB in Dragon immediately gives a DKG with subcubic communication and sublinear rounds.
3. *From subcubic to near-quadratic communication.* Finally, we apply the grouping and PCSBB methodology recursively. The recursion simultaneously generates the endemic coins required inside PCSBB, and yields a group-element DKG with $O(\kappa\lambda n^2 \log n)$ communication and $O(\kappa \log n)$ rounds. The DKG itself gives a single-shot common coin with the same complexities. It also supplies the threshold setup for a one-round common coin with quadratic communication that can be invoked repeatedly.

We call this technique suite *Circular Dragon*: it extends the Dragon methodology [46] from subcubic communication to sublinear rounds by breaking the circular dependence between round-efficient consensus and the common coin it requires.

With this more efficient common coin, we give two nontrivial applications. Parallel broadcast (PBC) and field-element DKG are independent, established problems; for each, we present a more efficient coin-aided construction.

Setting and assumptions. We consider a synchronous network and a strongly adaptive adversary corrupting $t < n/2$ parties. We use standard PKI and transparent public parameters (including a URS), as specified in § 3; each party registers its public key before the protocol begins. The main group-element DKG and common coin require neither a trusted dealer nor memory erasures. The concrete adaptive-security proof additionally uses the SXDH and co-OMDL assumptions in the algebraic group model and the random-oracle model.

2.1 Stage 1: Dragon reduces cubic communication to subcubic

A PVSS-based illustration of conventional DKG. We use publicly verifiable secret sharing (PVSS) to illustrate the source of the traditional costs. A PVSS dealer samples a degree-t polynomial f whose constant term $f(0)$ is the shared secret. It encrypts $f(j)$ for each party P_j and publishes a proof that the ciphertexts contain evaluations of one degree-t polynomial. Anyone can verify the resulting *PVSS transcript*, while only P_j can decrypt the share $f(j)$ intended for it [57]. A transcript for n recipients has $O(\lambda n)$ bits.

In a PVSS-based DKG, every party P_i acts as a dealer: it samples f_i and produces a transcript T_i. The parties must agree on a common set Qual of valid transcripts. Party P_j then outputs $F(j) = \sum_{i \in \mathsf{Qual}} f_i(j)$, $F = \sum_{i \in \mathsf{Qual}} f_i$, so $F(0)$ is the distributed secret.

The direct way to agree on Qual is to let every party Byzantine broadcast its transcript to the full network [42, 66, 67]. However, broadcasting one $O(\lambda n)$-bit transcript costs $O(\lambda n^2)$ bits, and doing so for all n dealers costs $O(\lambda n^3)$ bits. Moreover, the deterministic broadcast protocols used here have linear worst-case round complexity [41, 49]. Thus, the direct construction has two separate bottlenecks: cubic communication and linear rounds.

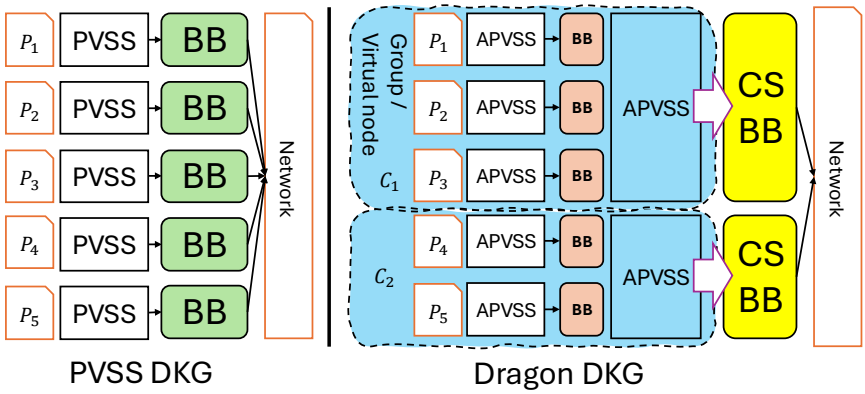

Fig. 1. Conventional PVSS-based DKG (left) and the Dragon grouping method (right). Dragon aggregates transcripts inside each consortium and disseminates only the consortium aggregates to the full network.

Dragon replaces many global dealers with a few consortium dealers. Dragon [46] (Fig. 1) reduces communication through three ideas that will also be important in our construction.

1. *Deterministic grouping.* The network is partitioned, according to a fixed public rule, into a small number m of disjoint groups, called *consortia*. This step requires neither random committee selection nor a trusted setup. Importantly, following the Pigeonhole Principle, *at least one consortium* has an honest majority.
2. *Local generation and aggregation.* Every party still produces a transcript whose encrypted shares are intended for the full network, but it disseminates that transcript only inside its own consortium. Consortium members agree on the valid local transcripts and aggregate them. This step uses aggregatable PVSS (APVSS) [18, 60]: transcripts sharing polynomials $f_1, \ldots, f_s$ can be combined into one transcript sharing $f_1 + \cdots + f_s$, without increasing the transcript size asymptotically. A consortium therefore produces one $O(\lambda n)$-bit aggregate rather than exposing every member's transcript to the full network.

3. *Consortium-to-network broadcast.* Each consortium acts as one virtual dealer and broadcasts its aggregate to all n parties using consortium-sender Byzantine broadcast (CSBB) [46]. In CSBB, a subset of parties jointly plays the role of the sender. If the sender consortium has an honest majority and its honest members hold the same message, every honest receiver delivers that message. Even for a corrupted-majority consortium, honest receivers agree on a consistent output. Most importantly, broadcasting an ℓ-bit message costs $\widetilde{O}(n\ell + \lambda n^2)$ bits, asymptotically the cost of an ordinary single-sender Byzantine broadcast rather than one broadcast per consortium member.

The above approach preserves secrecy, as honestly generated PVSS transcripts from the honest-majority consortium contribute to the final secret. It reduces the communication costs, as fewer global broadcasts are needed; as in [46], with $m = \sqrt{n}$ consortia, the overall cost is $\widetilde{O}(n^{2.5})$.

The remaining linear-round bottleneck creates a circularity. Although the m CSBB instances can run in parallel, each instance relies on deterministic Byzantine agreement across the entire network and therefore retains linear worst-case latency. A natural idea is to substitute a sublinear-round randomized consensus protocol. However, existing randomized consensus protocols at resilience $t < n/2$ require a (weak) common coin [63, 70], whose efficient instantiation is our exact goal. We are therefore left with the following challenge:

Can the consortium aggregates be broadcast in sublinear rounds while preserving Dragon's subcubic communication and without assuming a global common coin?

2.2 Stage 2: PCSBB uses one honest-majority consortium to obtain sublinear rounds

We begin with a simple observation. If a sender is known to be honest, broadcast is easy: the sender sends its message to everyone in one synchronous round. Byzantine broadcast needs agreement precisely because the sender may be faulty. Dragon does not identify which sender consortium has an honest majority, but the pigeonhole argument guarantees that at least one does. This motivates our new primitive, parallel consortium-sender Byzantine broadcast (PCSBB), which coordinates the consortium broadcasts so that one honest-majority sender consortium can synchronize the corresponding outputs.

PCSBB functionality: disseminating the consortium messages jointly. Consider a network $\mathcal{P} = \{P_1, \ldots, P_n\}$ and m sender consortia $\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(m)}$, each of which is a subset of $\mathcal{P}$. Consortium $\mathcal{D}^{(i)}$ has a message msg_i to disseminate, in the sense that all honest members of $\mathcal{D}^{(i)}$ hold the same message. All parties in $\mathcal{P}$ are receivers.

At the end of one PCSBB execution, every honest receiver outputs the same vector $\mathcal{V} = (v_1, \ldots, v_m)$. The guarantee for an individual entry depends on the corresponding sender consortium:

- If $\mathcal{D}^{(i)}$ has an honest majority and its honest members hold the same input msg_i, then $v_i = \mathsf{msg}_i$.
- Otherwise, PCSBB does not promise that v_i equals the consortium's prescribed input. It promises only that honest receivers agree on the same admissible value for that position (possibly a designated default).

PCSBB can therefore be placed directly into the Dragon framework. Each consortium first aggregates the valid APVSS transcripts produced by its members, and the m consortia then use one PCSBB execution to disseminate their aggregates together. Every honest party receives the same vector and hence aggregates the same valid entries into its DKG output. In addition, the vector contains the correct aggregate of every honest-majority consortium, including the honest contribution needed for DKG secrecy.

A rotating leader lets one honest sender synchronize all broadcasts. Our construction draws on rotating-leader state-machine replication (SMR) protocols [8, 22]. SMR allows parties to maintain the same ordered log despite Byzantine faults. Time is divided into *epochs*, and the leader of each epoch proposes a block extending the log. Votes from sufficiently many parties certify a block. A faulty leader may equivocate or stall, but the voting rules prevent honest parties from certifying incompatible histories. When an honest leader acts, it learns and extends the latest certified block, thereby bringing the honest parties onto one certified history. The property we need is that this honest leader synchronizes not only its own proposal, but also the certified proposals of earlier, possibly faulty, leaders.

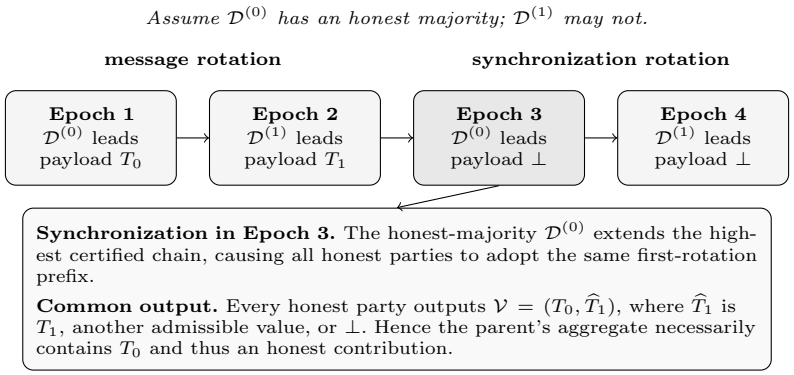

Fig. 2. A two-consortium PCSBB execution. The first rotation proposes input messages; the honest-majority leader in the second rotation synchronizes the resulting prefix.

This property jointly realizes m Byzantine broadcasts with distinct senders, provided that at least one sender is honest. Run two complete leader rotations, so that each sender leads twice and the execution has $2m$ epochs. During epochs $1, \ldots, m$, sender i proposes msg_i; during epochs $m+1, \ldots, 2m$, every sender proposes a dummy value. The second rotation is crucial: after all first-rotation messages have been proposed, it guarantees an epoch with an honest leader that synchronizes the resulting history. All honest parties can then extract the same m-entry vector. The entry of every honest sender equals its prescribed message, whereas an entry belonging to a corrupted sender need only be consistent.

PCSBB applies this observation by having each sender consortium emulate one SMR leader. Figure 2 illustrates the smallest nontrivial execution and, in particular, the role of the second leader rotation.

How a consortium efficiently emulates one SMR leader. An ordinary SMR leader performs three tasks: (i) choose a recent certified block to extend, (ii) sign the new block, and (iii) disseminate it to the network. Replacing one leader with a consortium makes each task nontrivial. For task (ii), we use a multiverse threshold signature (MTS) scheme [12], which lets the consortium produce one compact signature on the block. For task (iii), we reuse the communication-efficient consortium-sender delivery method from CSBB [46].

Task (i) is the conceptual difficulty. Consortium members may begin an epoch with different certificates, so they must reconcile their views before choosing one block to extend. We use an intra-consortium instance of multi-valued validated Byzantine agreement (MVBA): the members agree on one proposal, and a public validity rule ensures that the selected proposal contains a sufficiently recent certified block [10]. The full construction appears in § 4.2.

This MVBA must be randomized to obtain sublinear rounds, which is exactly where the consortium's endemic coin is used. Randomized agreement has an expected running time rather than a deterministic termination round, whereas PCSBB must move past an epoch if its corrupted leader consortium stalls. We therefore allocate $O(\kappa)$ rounds to each intra-consortium agreement. An honest-majority consortium completes within this deadline except with probability $2^{-\kappa}$; after the deadline, PCSBB advances to the next epoch regardless. Consequently, the $2m$ epochs take $O(\kappa m)$ rounds.

A first consequence: subcubic communication and sublinear rounds. Replacing Dragon's parallel CSBB instances with one PCSBB instance immediately yields a DKG with $\widetilde{O}(\kappa\lambda n^{2.5})$ communication and $O(\kappa\sqrt{n})$ rounds when the network is divided into $m = \sqrt{n}$ consortia of size $\sqrt{n}$. This construction needs an endemic coin in each consortium. At this single grouping level, a conventional intra-consortium DKG is already cheap enough: each instance costs $O(\lambda(\sqrt{n})^3)$ bits, so all $\sqrt{n}$ instances together cost $O(\lambda n^2)$ bits and do not dominate the overall communication. PCSBB therefore removes Dragon's linear-round bottleneck without sacrificing its subcubic communication.

2.3 Stage 3: recursive grouping gives near-quadratic communication and logarithmic rounds

A natural next step is to apply the grouping strategy recursively. The DKG of Bacho et al. [16] shows that recursive grouping can reduce the communication to near-quadratic. Our remaining challenge is to obtain logarithmic rounds as well: can PCSBB replace CSBB at every recursion level without making the endemic coins prohibitively expensive? We first recall the recursive structure and then explain how that structure itself generates the required coins.

Bacho et al.'s DKG has a multi-layer grouping structure. The recursive DKG of Bacho et al. [16] and Dragon were developed independently and concurrently. Nevertheless, they share a useful structural analogy. Dragon uses one grouping layer: small consortia create local APVSS aggregates and disseminate them to the full network through CSBB. Bacho et al. use multiple grouping layers. They first divide the full network into two consortia, each of which must produce an APVSS aggregate for the full network. Instead of producing this aggregate directly, each consortium is divided again: its two children produce smaller aggregates and disseminate them to their parent. This process continues until the consortia are small enough to use a conventional protocol. Thus, the same local-aggregation and consortium-broadcast pattern appears at every internal node of the recursion.

This viewpoint explains where PCSBB fits. Bacho et al. use CSBB to disseminate the two child aggregates at every internal node. We replace these two CSBB instances with one two-sender PCSBB instance. The replacement reduces the latency of each recursive layer while preserving the communication saving of the binary recursion.

The recursion must also provide the coins used by PCSBB. This replacement creates an apparent circularity. A PCSBB at a parent requires endemic coins inside its two child consortia. For children near the root, which may contain about $n/2$ parties, generating these coins with a conventional DKG would forfeit the desired complexity. Our key observation is that a recursive call can simultaneously produce (i) the aggregate that it must send to its parent and (ii) a DKG transcript for the consortium executing that call. The second output establishes the endemic coin that the consortium will use one level later.

To make this bottom-up computation concrete, consider a recursion of depth two:

$$\mathcal{P} \longrightarrow \{\mathcal{A}, \mathcal{B}\} \longrightarrow \{\mathcal{A}_0, \mathcal{A}_1, \mathcal{B}_0, \mathcal{B}_1\}.$$

A transcript *targeted to* a consortium encrypts one secret share for each member of that consortium. Aggregating transcripts with the same target therefore produces a DKG transcript for that target consortium.

1. *Leaf level.* Each party in leaf $\mathcal{A}_0$, for example, generates three APVSS transcripts: one targeted to $\mathcal{A}_0$, one targeted to $\mathcal{A}$, and one targeted to the full network $\mathcal{P}$. The members of $\mathcal{A}_0$ use a conventional protocol to agree on the valid contributions and aggregate transcripts with the same target. The aggregate targeted to $\mathcal{A}_0$ gives its members the local key shares needed for an endemic coin. The two aggregates targeted to $\mathcal{A}$ and $\mathcal{P}$ are carried to the next level. The other three leaves perform the same computation in parallel.
2. *Intermediate level.* The children $\mathcal{A}_0$ and $\mathcal{A}_1$ already have their own endemic coins. They use one PCSBB instance to disseminate their two remaining aggregates to all members of $\mathcal{A}$. The members of $\mathcal{A}$ aggregate the valid entries targeted to $\mathcal{A}$; the result establishes an endemic coin for $\mathcal{A}$. They separately aggregate the entries targeted to $\mathcal{P}$, obtaining one transcript $T_{\mathcal{A}\to\mathcal{P}}$ to carry to the root. Consortium $\mathcal{B}$ performs the analogous computation in parallel.
3. *Root level.* Consortia $\mathcal{A}$ and $\mathcal{B}$ now have the endemic coins needed for a final PCSBB. They use it to disseminate $T_{\mathcal{A}\to\mathcal{P}}$ and $T_{\mathcal{B}\to\mathcal{P}}$ to the full network. All honest parties obtain the same vector of valid transcripts and aggregate it into the global DKG output. This output supplies the threshold setup for the global common coin.

The same pattern extends to depth $d = O(\log n)$. A party at a leaf generates one transcript for each consortium on its leaf-to-root path. The target sizes decrease geometrically, so the total size generated by one leaf party is

$$O\Big(\lambda n + \lambda\frac{n}{2} + \cdots + \lambda\frac{n}{2^d}\Big) = O(\lambda n).$$

At each level, a consortium uses the transcript targeted to itself to establish its endemic coin and sends only the ancestor-targeted aggregates upward. Consequently, a coin is used only after the lower-level recursive call that establishes its setup has finished; there is no cyclic run-time dependency.

One honest-majority path carries secrecy to the root. Because a parent with an honest majority cannot have two corrupted-majority children, the recursion tree contains a root-to-leaf path on which every consortium has an honest majority. In the toy example, suppose this path is $\mathcal{P} \supset \mathcal{A} \supset \mathcal{A}_0$. The leaf aggregate from $\mathcal{A}_0$ contains a contribution unknown to the adversary. PCSBB validity ensures that this aggregate is included at $\mathcal{A}$, and then that $\mathcal{A}$'s aggregate is included at $\mathcal{P}$. The honest contribution therefore survives every aggregation step. At the root, all honest parties agree on the same DKG transcript, and its secret contains randomness unknown to the adversary. Section 5 formalizes this invariant and the composition of the underlying subprotocols.

The DKG output directly gives a single-shot common coin, so generating one coin from scratch has communication complexity $O(\kappa\lambda n^2 \log n)$ and round complexity $O(\kappa \log n)$. The same DKG also establishes the threshold setup required by the one-round common coin of Bacho et al. [16]. After this one-time setup, an application that needs many coins can invoke that one-round protocol repeatedly.

2.4 From the common coin to PBC and field-element DKG

Parallel broadcast (PBC) and field-element DKG are two independent, established problems. In PBC, every party has an input and all honest parties must agree on a vector containing the inputs of all honest senders. In field-element DKG, the parties generate Shamir shares of a secret scalar together with the corresponding public key, as required by standard discrete-log-based threshold schemes.

If efficient common coins are available, there are simple solutions for improving their efficiency. For PBC, it can select a small set of parties whose collected input vectors suffice to cover all honest inputs. For field-element DKG, it can select a small set of dealers whose valid PVSS transcripts suffice to hide the final secret. In both cases, selecting $O(\kappa)$ parties rather than broadcasting data from all n parties is the source of the communication saving.

Selection must occur only after the data are locked. On the other hand, strong adaptivity prevents us from simply selecting the $O(\kappa)$ parties and then asking them to broadcast. The adversary learns the selected indices and can corrupt those parties before their messages become irrevocably available. Enlarging the selection to $t + 1$ prevents the adversary from corrupting every selected party, but restores $O(n)$ broadcasts and cubic communication. We therefore introduce subset Byzantine broadcast (SBC) to lock every party's data before the common coin selects which inputs will be delivered.

SBC: disperse first, select second. SBC reverses the order. It starts with a propose phase, where every party erasure-encodes its input into n fragments, authenticates the fragments with an accumulator, and sends one fragment and witness to each recipient. An honest input is now recoverable from the fragments held by honest parties. Only after dispersal does the common coin choose a subset $\mathbb{S}$ of indices. Finally, in the deliver phase, parties relay fragments only for indices in $\mathbb{S}$, reconstruct those inputs, and run Byzantine agreement on each reconstructed value to obtain a common result.

The protocol pays the long-message reconstruction cost only for the selected inputs. More importantly, corrupting a selected sender after the propose phase cannot erase fragments already held by honest recipients.

PBC from SBC. Figure 3 illustrates this construction. In PBC, each party P_i begins with an input v_i. First, every party multicasts its signed input. Party P_j collects the valid signed inputs it receives into a vector V_j and proposes V_j to SBC. After all vectors are dispersed, the common coin selects $O(\kappa)$ indices and SBC delivers the corresponding vectors. With overwhelming probability, one selected vector was formed by an honest party and therefore contains all honest senders' inputs. All parties apply the same verification and deduplication rule to the delivered vectors, so they obtain the same output and include every honest input.

The PBC's expected communication cost is $O(\kappa n^2 \ell + \kappa\lambda n^2 \log n)$ and round complexity $O(\kappa \log n)$ when the common-coin setup is included. This construction is not merely a black-box replacement of deterministic

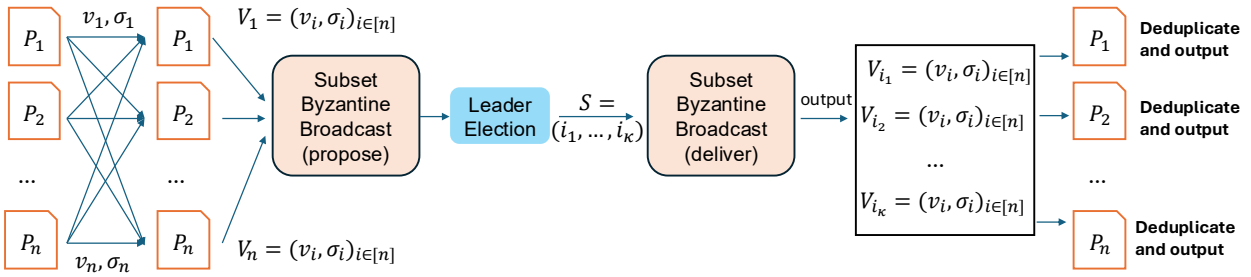

Fig. 3. PBC from SBC. Each party first multicasts its signed input and proposes the resulting vector to SBC. Selection occurs only after all vectors have been dispersed.

agreement by a randomized one: the disperse-before-select structure is what prevents after-the-fact removal under strong adaptivity. Section 6 gives the formal protocol.

Field-element DKG from SBC. As we discussed earlier, for the security of a DKG scheme, we only need one honest secret contribution. Therefore, instead of consuming $O(n)$ PVSS transcripts, $O(\kappa)$ randomly sampled transcripts suffice. Following the same intuition as our PBC protocol, we utilize the SBC component to allow the network to disperse $O(n)$ PVSS transcripts, while only $O(\kappa)$ randomly sampled transcripts are later delivered to all parties. As we do not need to aggregate the PVSS transcripts in this approach, we can employ instantiations that are compatible with field-element secrets. The full construction and proof appear in Section 7.

3 Preliminaries and Problem Formulation

3.1 System Model

Notation. Let λ and κ denote the cryptographic and statistical security parameters, respectively, with n polynomial in these parameters. We write $[i, n] = \{i, \ldots, n\}$ for $i < n$, abbreviate $[1, n]$ as $[n]$, and use $\mathsf{negl}(\lambda)$ for negligible quantities. The notations $x \leftarrow\!\!\$\ \mathbb{X}$ and $x \leftarrow X$ denote uniform sampling from a set and sampling from a distribution, respectively. We write $A(x; r)$ for execution with random tape r, and $y \leftarrow A(x)$ when the tape is sampled implicitly. An execution with identifier sid and inputs $(v_i)_{i\in[n]}$ is denoted $\Pi[\mathsf{sid}]\langle\{P_i(v_i)\}_{i\in[n]}\rangle$.

Communication model. We consider a synchronous, fully connected network $\mathcal{P} = \{P_i\}_{i\in[n]}$. Messages between uncorrupted parties are confidential, authenticated and delivered at the start of the next round. Communication complexity counts the bits sent by honest parties.

Threat model. We assume standard PKI and a uniform random string (URS) for transparent public parameters, with all public keys and parameters globally available. While the underlying tools may require group generators or random group elements as their setup parameters, those can be derived from the URS. In addition, as our results are in the random oracle model, the URS can also be obtained through a public random oracle call, without requiring a trusted dealer.

A PPT adversary $\mathcal{A}$ adaptively corrupts at most $t < n/2$ parties during setup and execution, learns their local states, and controls their subsequent behavior. It is rushing and may perform *after-the-fact removal*: after observing a round's messages, it can corrupt a sender and retract that sender's messages from the same round. Such an adversary is called *strongly adaptive*.

3.2 Common Coin

A common coin provides a shared, unpredictable, and bias-resistant random value. We consider a reusable setup followed by one-round coin generation [27].

SYNTAX. $\mathsf{Setup}(1^\lambda, n, t)$ includes an interactive DKG and any auxiliary setup. It produces a public key pk, public key shares $(pk_i)_{i\in[n]}$, and, for each P_i, a secret share sk_i and auxiliary setup view sv_i. In $\mathsf{Coin}^V[\mathsf{sid}]$, each P_i uses these outputs to multicast one message and computes $r_{\mathsf{sid}} \in V$ from the received messages.

PROPERTIES. Against the above adversary, we require the following properties, except with negligible probability.

Availability. All honest parties complete setup with a common public key and corresponding secret shares. Whenever all honest parties are activated for an instance sid, each outputs a value in V. *Agreement.* Any two honest outputs for the same sid coincide.

Bias-resistance and unpredictability. Consider polynomially many concurrent instances with distinct session identifiers. At time τ, let $\mathcal{C}(\tau)$ contain all parties corrupted so far, including initially corrupted parties, and let $\mathcal{A}_{\mathsf{sid}}(\tau)$ contain the distinct parties activated while honest to generate their message for sid. Activations remain recorded even if their messages are retracted. Put $a_{\mathsf{sid}}(\tau) = |\mathcal{A}_{\mathsf{sid}}(\tau) \setminus \mathcal{C}(\tau)|$.

For any adversarially chosen stopping time τ and target instance sid satisfying $|\mathcal{C}(\tau)| \le t$ and $a_{\mathsf{sid}}(\tau) < t + 1 - |\mathcal{C}(\tau)|$, let $\mathsf{state}_{\mathcal{A}}^{\mathsf{before}}$ be the adversary's state and r_{sid} the value obtained by honest completion of the instance. For every PPT distinguisher $\mathcal{D}$ and independent $u \leftarrow\!\!\$\, V$, we require

$$\left|\Pr[\mathcal{D}(\mathsf{state}_{\mathcal{A}}^{\mathsf{before}}, r_{\mathsf{sid}}) = 1] - \Pr[\mathcal{D}(\mathsf{state}_{\mathcal{A}}^{\mathsf{before}}, u) = 1]\right| \le \mathsf{negl}(\lambda).$$

The distinguisher may query the random oracles in the ROM but receives no further protocol messages or corruption responses.

EFFICIENCY GOALS. We aim to generate a common coin from scratch with $\widetilde{O}(n^2)$ communication complexity and $\widetilde{O}(1)$ round complexity. In the above syntax, this means that the combined cost of Setup protocols (including DKG) and a single invocation of $\mathsf{Coin}^V[\mathsf{sid}]$ is $\widetilde{O}(n^2)$ in communication and $\widetilde{O}(1)$ in rounds.

3.3 Cryptographic Building Blocks

Multiverse threshold signature with transparent setup. Threshold signature scheme allows a group of nodes to jointly produce a signature for a message which is valid under the public key of the group. Normally, a threshold signature scheme needs some sort of setup for every group, and each node needs to maintain distinct secret keys for all groups it is involved in. A multiverse threshold signature (MTS) scheme [19,37,53], instead, allows participants to jointly sign messages on behalf of different groups (called *universes* in the context), while each node only needs to keep one secret key. In an MTS scheme with a transparent setup [12], there is only a global PKI setup where all participants generate their own key pairs and publish their public keys.

Formally, we describe an MTS scheme by the following algorithms.

- $pp \leftarrow \mathsf{Setup}(1^\lambda)$. It generates the global public parameter pp which is an (implicit) input to all the subsequent algorithms. The setup is considered transparent when pp can be derived from uniformly distributed strings.
- $(vk_i, sk_i) \leftarrow \mathsf{KeyGen}(pp)$. Each user P_i runs this algorithm to generate its key pair (vk_i, sk_i).
- $(\mathsf{CK}_U, \mathsf{VK}_U) \leftarrow \mathsf{MTS.UniverseSetup}(\{vk_i\}_{i\in U}, k)$: It is a *deterministic* algorithm which computes a combine key CK_U and a verification key VK_U, according to the public keys of users in U, and the threshold k.
- $\sigma_i \leftarrow \mathsf{PartSign}(sk_i, \mathtt{msg})$. P_i can sign a message $\mathtt{msg}$ with its signing secret key sk_i and produce a partial signature σ_i.
- $\mathsf{true}(1)/\mathsf{false}(0) \leftarrow \mathsf{PartVrfy}(vk_i, \sigma_i, \mathtt{msg})$: This algorithm on inputs a message msg, a partial signature σ_i, and a public key vk_i verifies the partial signature σ_i.
- $\sigma \leftarrow \mathsf{Combine}(\mathsf{CK}_U, \{(i, \sigma_i)\}_{i\in S\subseteq U}, \mathtt{msg})$: Given the combine key CK_U, and a set of signatures $\{(i, \sigma_i)\}_{i\in S\subseteq U}$ for a message $\mathtt{msg}$, the algorithm outputs a succinct signature σ.
- $\mathsf{true}(1)/\mathsf{false}(0) \leftarrow \mathsf{Vrfy}(\mathsf{VK}_U, \sigma, \mathtt{msg})$: This algorithm verifies the signature σ for $\mathtt{msg}$ under the universe key VK_U. It returns 1, indicating that at least k nodes in U have signed the message; Otherwise, it returns 0.

For security, we consider the robustness and unforgeability. Both properties use a global setup establishing pp and a registered PKI. Write

$$\mathsf{MTS.Setup}^{\mathcal{A}}(1^\lambda) \longrightarrow (pp, (vk_i)_{i\in\mathcal{P}}, (sk_i)_{i\in\mathcal{H}}, \mathsf{state}_{\mathcal{A}}),$$

where $\mathcal{P} = [n]$ is the set of party indices and $\mathcal{H}$ contains the parties honest after initial corruption. The adversary's setup state includes the keys of initially corrupted parties. The keys CK_U and VK_U implicitly depend on both U and its threshold k.

Robustness. An MTS scheme is robust, if the following conditions are satisfied.

- A partial signature produced by an honest node is always valid. Formally, let sk_i and vk_i be the secret key and the public key of an honest node P_i, then for any message $\mathtt{msg}$, it holds that

$$\mathsf{PartVrfy}(vk_i, \mathsf{PartSign}(sk_i, \mathtt{msg}), \mathtt{msg}) = 1.$$

- Combining any k valid partial signatures by distinct nodes within the universe U yields a valid threshold signature for that message under VK_U. Formally, for any $U \subset \mathcal{P}$, let $(\mathsf{CK}_U, \mathsf{VK}_U) \leftarrow \mathsf{UniverseSetup}(\{vk_i\}_{i\in U}, k)$. For any set $\{(\sigma_j)\}_{j\in S}$ such that $\mathsf{PartVrfy}(vk_j, \sigma_j, \mathtt{msg}) = 1$ for every $j \in S \subset U$ and $|S| = k$, it holds that

$$\mathsf{Vrfy}(\mathsf{VK}_U, \mathsf{Combine}(\mathsf{CK}_U, \{(i, \sigma_i)\}_{i\in S}, \mathtt{msg}), \mathtt{msg}) = 1.$$

Unforgeability. For every PPT adversary $\mathcal{A}$, the following experiment returns 1 with probability at most $\mathsf{negl}(\lambda)$.

1. Run the global setup above. Initialize $C = \mathcal{P} \setminus \mathcal{H}$ and $V_m = \emptyset$ for every message m. Reject if $|C| > t$.
2. Give $\mathcal{A}$ the public setup and its setup state. A signing query (i, m) for $i \in \mathcal{P} \setminus C$ returns $\mathsf{PartSign}(sk_i, m)$ and adds i to V_m. A corruption query for a new index $i \in \mathcal{P} \setminus C$ adds i to C and returns sk_i, provided $|C| < t$; otherwise reject the query. Repeated corruption queries return the already revealed key. Invalid indices are rejected.
3. Let $\mathcal{A}$ output $(U^*, k^*, m^*, \sigma^*)$. Reject unless $\emptyset \neq U^* \subseteq \mathcal{P}$ and $1 \leq k^* \leq |U^*|$. Derive the target keys from the registered public keys, in canonical index order:

$$(\mathsf{CK}^*, \mathsf{VK}^*) \leftarrow \mathsf{UniverseSetup}((vk_i)_{i\in U^*}, k^*).$$

4. Return 1 exactly when

$$\mathsf{Vrfy}(\mathsf{VK}^*, \sigma^*, m^*) = 1 \quad \text{and} \quad |U^* \cap (V_{m^*} \cup C)| < k^*.$$

Thus only distinct signers in the target universe count. Repeated signing queries, signatures by parties outside U^*, and signatures on other messages do not increase the available signer count. A signer queried before corruption is counted only once.

Instantiation. Throughout the paper, we instantiate MTS by the threshold signature scheme (TSS) in [12, Section 5] with PKI assumption and a transparent setup. This instantiation gives threshold signatures of size $O(\lambda \log |U|)$, where U is the universe of signers. The computational overhead of $\mathsf{PartSign}$ and $\mathsf{PartVrfy}$ is $O(1)$, inherited from the BLS digital signature scheme, while $\mathsf{Combine}$ and Vrfy inherit $O(|U|)$ computational cost from [12].

For this instantiation, the combine and verification keys can be represented by the public descriptor $(U, k, (vk_i)_{i\in U})$, with indices in canonical order. The descriptor binds both the universe and its threshold and is derived locally from the registered PKI; it requires no additional secret setup.

Cryptographic accumulators. A cryptographic accumulator provides a succinct representation of a set while ensuring efficient membership verification. Formally, such an accumulator scheme, denoted as Acc, comprises the following four algorithms: (1) $\mathsf{Gen}(1^\lambda, n)$ outputs an accumulator key ak; (2) $\mathsf{Eval}(ak, \mathcal{S})$ on

inputs ak and a set $\mathcal{S}$ to be accumulated, it returns an accumulated value u for the set $\mathcal{S}$; (3) $\mathsf{Wit}(ak, u, \mathcal{S}, s_i)$ on inputs ak, u for the set $\mathcal{S}$, and an element $s_i \in \mathcal{S}$, it returns a membership witness w_i for s_i; (4) $\mathsf{Vrfy}(ak, u, s_i, w_i)$ returns a bit representing whether s_i is indeed accumulated into u ("1") or not ("0") according to the given witness w_i.

An accumulator scheme is said to be *correct*, if for $ak \leftarrow \mathsf{Gen}(1^\lambda, n)$, any set $\mathcal{S} = \{s_i\}_{i\in[n]}$, $u \leftarrow \mathsf{Eval}(ak, \mathcal{S})$, and any $w_i \leftarrow \mathsf{Wit}(ak, u, \mathcal{S}, s_i)$, it holds that $\Pr[\mathsf{Vrfy}(ak, u, s_i, w_i) = 1] = 1$. An accumulator scheme is *unforgeable*, if for an honestly generated ak, and any PPT adversary,

$$\Pr\left[(\mathcal{S}, s^*, w^*) \leftarrow \mathcal{A}(ak) : s^* \notin \mathcal{S} \wedge \mathsf{Vrfy}(ak, \mathsf{Eval}(ak, \mathcal{S}), s^*, w^*) = 1\right] \leq \mathsf{negl}(\lambda).$$

For simplicity, throughout this paper, we consider an accumulator scheme whose Eval and Wit are deterministic.

Instantiation. We primarily consider the Merkle tree based accumulator, which solely requires a hash key as its setup, and its witness size is $O(\lambda \log n)$.

Erasure code scheme. Let Σ denote the fragment alphabet. A (k, n) erasure code (EC) scheme over Σ [23] consists of two deterministic algorithms Enc and Dec. The Enc algorithm maps any vector $\mathbf{m} = (m_1, \cdots, m_k)$ of k data fragments into a vector $\mathbf{c} = (\mathsf{c}_1, \ldots, \mathsf{c}_n)$ of n coded fragments, such that any k elements of the code vector $\mathbf{c}$ are enough to reconstruct $\mathbf{m}$ due to the Dec algorithm. I.e., for any $\mathbf{m} \in \Sigma^k$ and any $\mathbb{I} \subset [n]$ that $|\mathbb{I}| = k$, we have

$$\Pr[\mathsf{Dec}(\{(i, \mathsf{c}_i)\}_{i\in\mathbb{I}}) = \mathbf{m} \mid \mathbf{c} := (\mathsf{c}_1, \cdots, \mathsf{c}_n) \leftarrow \mathsf{Enc}(\mathbf{m})] = 1.$$

Instantiation. We use Reed–Solomon erasure codes, with the reconstruction threshold specified by each protocol. Encoding an ℓ-bit message with threshold k gives fragments of size $O(\ell/k + \log n)$, including padding to field symbols. Thus the total encoded size is $O(n\ell/k + n \log n)$. In SBC we choose $k = n - t > n/2$, so each fragment has size $O(\ell/n + \log n)$ for every $t < n/2$. No global assumption $n = 2t + 1$ is needed.

NIZK in the random oracle model. For a relation R, a non-interactive zero-knowledge argument of knowledge consists of $\mathsf{Prove}^H(x, w)$ and $\mathsf{Vrfy}^H(x, \pi)$, where H is a random oracle and $R(x, w) = 1$. Public group parameters are implicit. We require perfect completeness and adaptive, multi-theorem zero knowledge: polynomially many adaptively requested honest proofs can be replaced by proofs from a stateful simulator $\mathsf{SimProve}(x)$, without the witnesses, while the simulator consistently answers and programs H. The real and simulated views must be computationally indistinguishable. In this comparison, proof requests supply valid statement–witness pairs; only the statement is passed to the simulator.

Definition 1 (Weak simulation extractability in the ROM). *Fix the above simulator. A PPT adversary $\mathcal{A}$ has adaptive access to its random-oracle interface and to $\mathsf{SimProve}(x)$, which takes no witness. Let $\mathcal{T}$ record the returned statement–proof pairs. Define*

$$\mathsf{Fresh} := \left[\mathsf{Vrfy}^H(x^*, \pi^*) = 1 \ \wedge \ (x^*, \pi^*) \notin \mathcal{T}\right], \qquad \epsilon := \Pr[\mathsf{Fresh}],$$

where (x^*, π^*) is $\mathcal{A}$'s output and acceptance and freshness are evaluated using the initial run's oracle table and log. There must exist a polynomial-time extractor $\mathsf{Ext}^{\mathcal{A}}$, allowed to replay $\mathcal{A}$ with the same random tape and to control both oracle interfaces, such that, for a negligible extraction error ν, a polynomial p_0, and a constant $d_0 > 0$,

$$\Pr[\mathsf{Fresh} \ \wedge \ R(x^*, w^*) = 1] \ \geq \ \frac{(\epsilon - \nu)^{d_0}}{p_0(\lambda)} \quad \textit{whenever } \epsilon \geq \nu.$$

Here w^* is extracted for the statement in the initial run; the extractor receives that run's output and oracle logs. This is the rewinding notion of [43, Definition 6].

Freshness excludes only exact simulated pairs, so a new proof for a previously simulated statement remains eligible for extraction. Simulation soundness alone merely excludes accepted false statements and supplies no witness. Definition 1 supplies the extraction property needed in § 7; it does not assert online extraction or extraction from every individual accepting run. We use it in reductions: a non-negligible probability of an inconsistent accepted share yields a non-negligible probability of extracting an opening that violates commitment binding. Exact simulated pairs are handled separately.

3.4 Aggregatable PVSS

An (n, t)-secret sharing (SS) scheme allows a dealer to distribute a secret s among n participants. Any group of $t+1$ honest parties can reconstruct s, yet any smaller group (up to t parties) remains oblivious to s. Whereas SS assumes a trustworthy dealer, verifiable secret sharing (VSS) addresses the possibility of a malicious dealer by letting a receiver validate the consistency of its share with a public commitment. Publicly-verifiable secret sharing (PVSS) takes VSS a step further: all encrypted shares, together with a verifiable proof for the consistency of these encrypted shares, are placed on a public channel for universal validation. An *aggregatable* PVSS further can compress multiple PVSS transcripts into a single publicly verifiable transcript.

The Syntax. We describe aggregatable PVSS with the following eight algorithms/phases. For simplicity, we assume that the "native" transcripts (produced by Deal) and the aggregated transcripts are in the same form (though their contribution lists can have different lengths), and thus all algorithms and properties apply to both types of transcripts. The syntax and definitions for a (non-aggregatable) PVSS can be obtained by removing the algorithm Agg.

- $\mathsf{Init}(1^\lambda, n)$: In the initial phase, a CRS $\mathtt{crs}$ is generated, and the encryption/decryption keys $\{(ek_i, dk_i)\}_{i\in[n]}$ for all participants are set up. $\mathtt{crs}$ is an implicit input for all other algorithms.
- $\mathsf{Deal}((ek_i)_{i\in[n]}, t, \mathtt{cid}) \to (\mathsf{Trans}, sk)$: On inputs the public key list, the threshold t, and the CID of the creator, it produces a secret $sk \in \mathcal{SK}$ and a transcript Trans, consisting of a commitment com to the secret sk, ciphertexts $(c_i)_{i\in[n]}$, a proof π of validity, and a singleton contribution list recording the CID, constant commitment, and proof.
- $\mathsf{Agg}((\mathsf{Trans}_i)_{i\in[m]}, (ek_i)_{i\in[n]}) \to \mathsf{Trans}$: It aggregates an indexed list of transcripts and concatenates their contribution lists, preserving every occurrence, including repeated records and repeated input transcripts.
- $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}) \to b$: It checks if Trans is valid.
- $\mathsf{getCID}(\mathsf{Trans})$: It returns the set of CID labels occurring in the contribution list. This helper does not change the list or the multiplicities used by any product.
- $\mathsf{PubDriv}(\mathsf{Trans}) \to (pk, (pk_i)_{i\in[n]})$: It derives the public key (shares).
- $\mathsf{Dec}(ek_i, dk_i, \mathsf{Trans}) \to sk_i$: One can decrypt the ciphertext c_i in Trans and obtain the secret share sk_i.
- $\mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}}) \to sk$: It first determines coefficients $\{\alpha_i\}_{i\in\mathbb{I}}$, where $\alpha_i \in \mathbb{N}$ based on $\mathbb{I}$ and reconstructs the committed secret sk as $\bigoplus_{i\in\mathbb{I}} \alpha_i sk_i$ from any subset $\mathbb{I} \subset [n]$ and $|\mathbb{I}| = t+1$.

We use the APVSS construction of [60], with an explicit contribution list to specify aggregation when inputs overlap. Each record keeps its CID, constant commitment, and proof together; repeated records are retained. For this instantiation, the generic $\mathtt{crs}$ denotes public group parameters under the transparent setup of § 3; it carries no secret setup trapdoor. In terms of functionalities, the following algorithm in ours $\mathsf{Init}, \mathsf{Deal}, \mathsf{Agg}, \mathsf{PubVrfy}, \mathsf{Dec}, \mathsf{Rec}$ are equivalent to theirs $\mathsf{Keys}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{Ver}, \mathsf{Dec}, \mathsf{Rec}$ respectively. For the getCID and $\mathsf{PubDriv}$ algorithms, they are helper functions that extract information from a transcript directly and compute.

Building blocks. We first introduce a few needed cryptographic building blocks.

Pairing. There is an efficient deterministic algorithm $\mathsf{GroupGen}$ which outputs the description of the pairing groups, including $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of order p, g is the generator of $\mathbb{G}_1$, h is the generator of $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a bilinear map.

Signature of Knowledge. A signature of knowledge (SoK) scheme SoK for an NP language L in the random oracle model consists of two algorithms. $\mathsf{Sign}(x, w, m)$ on inputs a statement x, a witness w, and a message m, produces a signature σ on m. $\mathsf{Vrfy}(x, m, \sigma)$ verifies the signature. A SoK scheme should satisfy the Sim-Ext security [30]. The SoK in the APVSS scheme can be instantiated by using Schnorr signature.

- $\mathsf{Init}(1^\lambda, n)$. (1) $\mathsf{GroupGen} \to (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$; Define $\mathtt{crs} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$. (2) For uncorrupted $i \in [n]$, $dk_i \leftarrow\!\$\ \mathbb{Z}_p$, and $ek_i = h^{dk_i}$;
- $\mathsf{Deal}((ek_i)_{i\in[n]}, t, \mathtt{cid})$. Sample $(a_0, \ldots, a_t) \leftarrow\!\$\ \mathbb{Z}_p^{t+1}$ and define $f(X) = \sum_{k=0}^{t} a_k X^k$, $A_i = g^{f(i)}$ for $i \in [0, n]$, and $\hat{Y}_i = ek_i^{f(i)}$ for $i \in [n]$. Compute $\varsigma \leftarrow \mathsf{SoK.Sign}(A_0, f(0), \mathtt{cid})$ and the singleton list $\mathcal{R} = [(\mathtt{cid}, A_0, \varsigma)]$. Return $sk = h^{f(0)}$ and $\mathtt{Trans} = ((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \mathcal{R})$.

- $\mathsf{Agg}((\mathtt{Trans}_j)_{j\in[m]}, (ek_i)_{i\in[n]})$. All inputs use the same recipient keys and threshold. Parse $\mathtt{Trans}_j = ((A_i^{(j)})_{i\in[n]}, (\hat{Y}_i^{(j)})_{i\in[n]}, \mathcal{R}_j)$. Compute

$$A_i = \prod_{j=1}^{m} A_i^{(j)}, \qquad \hat{Y}_i = \prod_{j=1}^{m} \hat{Y}_i^{(j)}, \qquad \mathcal{R} = \mathcal{R}_1 \| \cdots \| \mathcal{R}_m,$$

  where $\|$ denotes list concatenation, including every repeated record. Return $\mathtt{Trans} = ((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \mathcal{R})$. Inputs are processed in their public sender or child index order, and each input list retains its order. No record is removed because its CID, commitment, or proof has occurred before.
- $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathtt{Trans})$. Parse $\mathtt{Trans} = ((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \mathcal{R})$, where $\mathcal{R} = [(\mathtt{cid}_a, C_a, \varsigma_a)]_{a=1}^{L}$. Reject malformed encodings, including elements outside their prescribed groups. Check $\mathsf{SoK.Vrfy}(C_a, \mathtt{cid}_a, \varsigma_a) = 1$ for every position $a \in [L]$. Assume $0 \le t < n < p$. After fixing the transcript, sample $q(X) = \sum_{k=0}^{n-t-1} q_k X^k$ with independent uniform coefficients $q_k \leftarrow\!\$\, \mathbb{Z}_p$, and compute

$$\mathsf{code}_i^{\perp} = \frac{q(i)}{\prod_{j=0, j\neq i}^{n}(i-j)}, \qquad i \in [0, n].$$

  Set $A_0 = \prod_{a=1}^{L} C_a$, with one factor for every list position, and check the low-degree condition of SCRAPE [29]:

$$\prod_{i=0}^{n} A_i^{\mathsf{code}_i^{\perp}} = 1. \tag{1}$$

  This test accepts every vector $(A_i = g^{f(i)})_{i\in[0,n]}$ with $\deg f \le t$, and accepts any fixed vector not of this form with probability exactly $1/p$ over the sampled q. Also check $e(g, \hat{Y}_i) = e(A_i, ek_i)$ for every $i \in [n]$. Return 1 exactly when all checks pass.
- $\mathsf{getCID}(\mathtt{Trans})$. Return $\{\mathtt{cid}_a : a \in [L]\}$, the support of the CID labels in $\mathcal{R}$.
- $\mathsf{PubDriv}(\mathtt{Trans})$. Return $pk = \prod_{a=1}^{L} C_a$ and $(pk_i = A_i)_{i\in[n]}$, retaining every repeated factor in the product.
- $\mathsf{Dec}(ek_i, dk_i, \mathtt{Trans})$. It returns $sk_i = \hat{Y}_i^{\frac{1}{dk_i}}$.
- $\mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}})$. It first computes the Lagrange coefficients $\{\lambda_i\}_{i\in\mathbb{I}}$ based on $\mathbb{I}$, and then $sk = \prod sk_i^{\lambda_i}$.

Multiplicities and size bounds. Write $\mathcal{R}(T)$ for the contribution list of T, $\mathsf{len}(T) = |\mathcal{R}(T)|$, and $\mathsf{mult}(R, T)$ for the number of occurrences of the exact record $R = (\mathtt{cid}, C, \varsigma)$ in that list. Records are explicit, rather than encoded using unrestricted integer weights. We write $\mathsf{PVSS.Vrfy}_B(T) = 1$ when $1 \le \mathsf{len}(T) \le B$ and the above public verification accepts; the relevant keys and threshold are implicit. Thus the bound is checked before processing the proofs. Each application specifies $B < p$. CID membership is only a label test: an honest contribution means an occurrence of the actual record generated by that honest party, not merely a record bearing its CID.

Lemma 1 (Aggregation with repeated contributions). *If $T_1, \ldots, T_m$ are well-formed, valid transcripts for the same recipient keys and threshold, then their aggregate is valid, even when their contribution lists overlap. Moreover,*

$$\mathsf{len}(\mathsf{Agg}((T_j)_{j=1}^{m})) = \sum_{j=1}^{m} \mathsf{len}(T_j),$$
$$\mathsf{mult}(R, \mathsf{Agg}((T_j)_{j=1}^{m})) = \sum_{j=1}^{m} \mathsf{mult}(R, T_j).$$

Proof. Let f_j be a degree-at-most-t polynomial underlying T_j. The aggregate share commitments encode $F = \sum_j f_j$, whose degree is at most t, and its encrypted shares are $ek_i^{F(i)}$. Because concatenation preserves

all record occurrences, the product of the aggregate's constant commitments is

$$\prod_{j=1}^{m} \prod_{a=1}^{|\mathcal{R}(T_j)|} C_{j,a} = \prod_{j=1}^{m} g^{f_j(0)} = g^{F(0)}.$$

Every attached proof remains valid. The length and multiplicity identities follow directly from concatenation. Consequently, adding further valid inputs cannot remove an existing record. Whenever the output length is less than p, the multiplicity of each retained record lies in $\{1, \ldots, p-1\}$ and is nonzero in $\mathbb{Z}_p$. When validity is established using the randomized low-degree tests, this argument is conditioned on no false acceptance; the error bound below accounts for that event.

Low-degree test. For completeness, let $D_i = \prod_{j=0, j\neq i}^{n}(i-j)$. If $\deg f \leq t$ and $\deg q \leq n-t-1$, then $\deg(fq) \leq n-1$. The Lagrange interpolation identity gives

$$\sum_{i=0}^{n} \frac{f(i)q(i)}{D_i} = [X^n](f(X)q(X)) = 0,$$

which proves that Eq. (1) always accepts an honestly generated transcript. For soundness, write $A_i = g^{a_i}$. The vectors $(q(i)/D_i)_{i=0}^{n}$, as q ranges over the polynomials of degree at most $n-t-1$, form the full $(n-t)$-dimensional dual of the degree-at-most-t evaluation code on $0, \ldots, n$. Thus, if $(a_0, \ldots, a_n)$ is outside that code, its inner product with a uniformly sampled dual vector is uniform in $\mathbb{Z}_p$, and is zero with probability exactly $1/p$. This bound also holds conditioned on any preceding history, provided the transcript is fixed before fresh test coefficients are sampled; a union bound gives failure probability at most Q/p for Q such tests.

3.5 Consensus Building Blocks

Multi-valued Validated Byzantine Agreement. In an (n, t, ℓ)-Multi-valued Validated Byzantine Agreement (MVBA) protocol, there are n parties $\{P_1, \ldots, P_n\}$, where each P_i has an ℓ-bit input v_i and decides an output value. MVBA is defined w.r.t. an external predicate Q on the input values. Against any adaptive PPT adversary $\mathcal{A}$ that corrupts up to t parties, a secure MVBA ensures the following properties with all but negligible probability:

- **Termination.** If each honest node P_i provides an input v_i such that $Q(v_i) = 1$, then every honest party would output a value.
- **External-Validity.** If an honest node outputs a value v, then $Q(v) = 1$.
- **Agreement.** The output messages of all honest parties would be the same.

Instantiation. We consider a variant of the synchronous MVBA protocol provided in [10], which requires $O(1)$ rounds and incurs the communication cost of $O(n\ell + \lambda n^2 \log n)$ bits in expectation, for agreeing on ℓ bits among n nodes. The underlying threshold signature scheme can be instantiated with the MTS scheme [12] where the size of a threshold signature is $O(\lambda \log n)$, and the underlying accumulator scheme is implemented using the hash-based Merkle tree. In addition, this MVBA assumes common coin subroutines which we will realize when applying the MVBA protocol.

Assuming the security of the underlying hash function and the common coin subroutine, an instance $\mathsf{MVBA}[\mathsf{sid}]$ of this MVBA protocol is secure if the adversary cannot forge an MTS signature for messages in this instance. Therefore, its security remains even when the adversary can query the signing oracle for messages which do not start with "sid", which helps prove the composition security of the MVBA protocol when every instance has been assigned with a unique identifier.

Termination deadlines. The expected constant-round bound does not itself give a deadline. We use the stronger progress property of the cited MVBA: with fresh session- and iteration-specific ideal common

coins, every unfinished iteration has probability at least 1/2 of making all honest parties terminate within a constant number of further rounds, conditioned on the preceding execution history. Proposals are fixed before the leader is revealed, and the certified-proposal mechanism protects progress against subsequent corruption [10, Section 5]. The following argument needs no independence between iterations or concurrent executions.

Lemma 2 (Consensus deadlines). *Consider at most $N \geq 1$ invocations of a consensus protocol with the conditional progress property above, each iteration costing $O(1)$ rounds. With ideal common coins and valid authentication, all invocations terminate within $O(\kappa + \log N)$ rounds of their respective starts, except with probability at most $2^{-\kappa}$. In particular, for $N \leq 2^{\kappa}$, a deadline $\Delta_{\mathsf{MVBA}} = O(\kappa)$ suffices for the cited MVBA. For a polynomially bounded execution with the cryptographic instantiations, add their joint failure or distinguishing advantage $\nu(\lambda) = \mathsf{negl}(\lambda)$ to the failure bound.*

Proof. Index invocations by their launch order, including adaptively launched invocations, and let X_i count iterations until successful progress in invocation i. Set $X_i = 0$ for unused indices. If $\mathcal{F}_{i,s}$ is the execution history before its next iteration, the progress property gives, on $X_i > s$,

$$\Pr[X_i > s + 1 \mid \mathcal{F}_{i,s}] \leq \tfrac{1}{2}.$$

Iterated conditioning yields $\Pr[X_i > L] \leq 2^{-L}$. A union bound, not a product of probabilities, therefore gives

$$\Pr[\exists i \leq N : X_i > L] \leq N 2^{-L}.$$

Take $L = \kappa + \lceil \log_2 N \rceil$; the right side is at most $2^{-\kappa}$. Constant-round iterations and the fixed setup and final notification rounds give the stated deadline. Replacing ideal components by their secure implementations changes this probability by at most $\nu(\lambda)$. □

Here N is a predetermined upper bound on the total number of relevant invocations, not merely those simultaneously active. All uses of an $O(\kappa)$ deadline assume $\log N = O(\kappa)$; otherwise the deadline is $O(\kappa + \log N)$. For example, when N is polynomial in n, the number Q of top-level invocations, and κ, it suffices that $\log(nQ) = O(\kappa)$. The ideal probability bound is valid for arbitrary N; the cryptographic claim is for polynomially bounded adversaries and executions.

Byzantine agreement. In an (n, t, ℓ)-Byzantine Agreement (BA) protocol, there are n parties $\{P_1, \ldots, P_n\}$, where each P_i has an ℓ-bit input v_i and decides an output, denoted as $\mathsf{BA}\langle\{P_i(v_i)\}\rangle$. Against any adaptive PPT adversary $\mathcal{A}$ that corrupts up to t parties, a secure BA ensures the following properties with all but negligible probability:

- **Validity.** If all honest parties share the same input v, they all output v.
- **Agreement.** The output messages of all honest parties would be the same.
- **Termination.** Every honest party would decide on an output message.

Instantiation. We consider the BA protocol implied by [7] and [67]. In particular, the BA protocol by Abraham et al. [7] achieves expected $O(1)$ round complexity and expected $O(n^2|\mathsf{ThldSig}|)$ communication complexity when the input size $\ell = O(\lambda)$, where $|\mathsf{ThldSig}|$ represents the signature size of the underlying threshold signature scheme. For cases where $\ell > \lambda$, Nayak et al. [67] proposed the state-of-the-art BA extension protocols. Throughout this paper, we consider the BA protocol obtained by applying Nayak et al.'s extension technique to Abraham et al.'s protocol, which achieves expected $O(1)$ round complexity and expected $O(n\ell + \lambda n^2 \log n + n^2|\mathsf{ThldSig}|)$ communication complexity.

The BA protocol from [7] assumes common coin subroutines which we will realize when applying the BA protocol. The underlying threshold signature scheme can be instantiated with the MTS scheme [12] where the size of a threshold signature is $O(\lambda \log n)$. The extension protocol proposed by Nayak et al. [67] requires an accumulator scheme and an Erasure code scheme, which can be instantiated by the hash-based Merkle tree

and the Erasure code scheme from [23], respectively. In summary, the expected communication complexity of the BA protocol becomes $O(n\ell + \lambda n^2 \log n)$.

Termination deadlines. The strongly adaptive BA of [7, Sections 3.3–3.4] prepares proposals before revealing each fresh leader. A leader honest through that preparation ensures progress even if subsequently corrupted. Thus its iterations satisfy the same conditional progress bound of at least $1/2$ with ideal common coins. The BA extension in [67, Figure 2] uses two sequential base-BA calls followed by constant-round dispersal and reconstruction, so Lemma 2 gives $\Delta_{\mathsf{BA}} = O(\kappa + \log N)$ for at most N extended-BA invocations, including any constant increase in the number of base calls. This is $O(\kappa)$ when $\log N = O(\kappa)$, with total deadline failure at most $2^{-\kappa} + \nu(\lambda)$ after allocating the failure budget across those calls.

Byzantine Broadcast. In an (n, t, ℓ)-Byzantine Broadcast (BB) protocol, there are n parties $\{P_1, \ldots, P_n\}$, and one of them is designated as the sender P_s which has ℓ-bit input v_s. Against any adaptive PPT adversary $\mathcal{A}$ that corrupts up to t parties, a secure BB ensures the following properties with all but negligible probability:

- **Validity.** If the sender is honest, all honest nodes output v_s.
- **Agreement.** The output messages of all honest parties would be the same.
- **Termination.** Every honest party would decide on an output message.

Instantiation. We consider the deterministic BB protocol implied by [66] and [67], which, under a transparent setup, requires $O(n)$ rounds and incurs the communication cost of $O(n\ell + \lambda n^2 \log n)$ bits for broadcasting ℓ bits to n nodes. In particular, the BB protocol in [66] assumes a threshold signature for reducing communication complexity, which can be realized using the MTS scheme from [12] which we discussed above. And the extension technique from [67] requires a cryptographic accumulator which can be implemented using the hash-based Merkle tree. An instance $\mathsf{BB}[\mathsf{sid}]$ of this BB protocol is secure as long as the adversary cannot forge MTS signatures for messages in this instance. Therefore, its security remains even when the adversary can query the signing oracle for messages which do not start with "sid", which grants the composition security of the BB protocol when every instance has been assigned with a unique identifier, as discussed in [35, 46, 65].

4 Parallel Consortium-Sender Byzantine Broadcast

This section introduces parallel consortium-sender Byzantine broadcast (PCSBB), our core consensus component towards realizing our more efficient DKG.

4.1 The Definition

Consortium-sender Byzantine broadcast (CSBB) [16, 46] allows a sender consortium to emulate a single sender when broadcasting to a network $\mathcal{P}$. If the consortium has an honest majority whose honest members share an input $\mathtt{msg}$, all receivers output $\mathtt{msg}$; otherwise, CSBB still guarantees agreement.

PCSBB generalizes CSBB to multiple sender consortiums. When the consortiums partition an honest-majority network, at least one has an honest majority; PCSBB exploits this fact to achieve better efficiency than running independent CSBB instances. We formalize the primitive below.

Definition 2. *A PCSBB protocol involves n participants $\mathcal{P} = \{P_1, \ldots, P_n\}$ and is paired with m consortium-sender groups $\{\mathcal{D}^{(i)}\}_{i\in[m]}$. A consortium-sender group is* **honest** *if it has an honest majority, and* **corrupted** *otherwise. A PCSBB protocol is (n, m, t)-secure if the following properties hold with overwhelming probability against any adaptive PPT adversary $\mathcal{A}$ corrupting up to t parties in $\mathcal{P}$:*

- ***Termination.*** *Every honest node outputs a vector $\mathcal{V} = (v_1, \ldots, v_m)$.*
- ***Agreement.*** *In the existence of an honest consortium-sender group whose honest participants also share the common input, all honest nodes in $\mathcal{P}$ output the same vector $\mathcal{V}$.*
- ***Validity.*** *If the honest nodes in an honest consortium $\mathcal{D}^{(i)}$ share the same input msg_i, then $v_i = \mathsf{msg}_i$.*

4.2 The Construction

Our PCSBB construction adapts the rotating-leader SMR paradigm [9, 22] to the setting where each leader is a sender consortium rather than a single party. As discussed in Section 2.2, the key is to let each consortium emulate an SMR leader using only its *endemic* common coin, thereby avoiding the need for a global common coin.

Building blocks. PCSBB uses the MTS scheme, the accumulator Acc and the erasure code EC of § 3.3, the MVBA protocol of § 3.5, and the data-delivery subroutine DD of § 4.4. We fix the following notation and parameters.

MTS. We write $(\mathsf{VK}_{\mathcal{P}}, \mathsf{CK}_{\mathcal{P}})$ and $(\mathsf{VK}_{\mathcal{D}_i}, \mathsf{CK}_{\mathcal{D}_i})$ for the universe keys of the whole network and of consortium $\mathcal{D}_i$, respectively. Threshold signatures have $O(\lambda \log |U|)$ bits.

Accumulator and erasure code. We use a Merkle-tree accumulator, with $O(\lambda)$-bit accumulator values and $O(\lambda \log n)$-bit witnesses, and write $\mathsf{EC.Encode}$ and $\mathsf{EC.Decode}$ for the encoding and decoding algorithms of a deterministic (k, n) erasure code with $k = \lfloor n/2 \rfloor + 1$.

MVBA. Each MVBA instance runs within a consortium of size η using the consortium's endemic common coin, and is truncated after $\Delta_{\mathsf{MVBA}} = O(\kappa)$ rounds. For the $O(\lambda \log n)$-bit inputs used by PCSBB, its communication is $O\big(\kappa(\eta^2 \lambda \log \eta + \eta \lambda \log n)\big)$ bits, also for corrupt-majority consortiums; an honest-majority consortium reaches agreement by the deadline except with the failure probability of Lemma 2.

Data Delivery (DD). $\mathsf{DD.send}[\mathsf{did}](M, u)$ disperses erasure-coded fragments of M committed by accumulator u, $\mathsf{DD.relay}[\mathsf{did}](u)$ relays the caller's valid fragment, and $\mathsf{DD.recover}[\mathsf{did}](u)$ reconstructs M. If an honest party sends M and all honest parties relay u, every honest party recovers the same M within two rounds, at a total cost of $O(n\ell + \lambda n^2 \log n)$ bits for an ℓ-bit message.

The formal description. We present PCSBB in Fig. 4. It requires a setup phase in which (1) the public parameters for the underlying MTS, MVBA and Acc are honestly generated and available to all nodes, (2) the PKI setup for the underlying MTS has been established, and (3) within each consortium $\mathcal{D}_i$ the setup for enabling one-round common coin has been established.

All signed tuples use canonical, injective encodings. We define

$$\mu_{\mathsf{lac}}(e, c) = \langle \mathsf{sid}, \mathsf{tag}_{\mathsf{PCSBB}}, \texttt{leader-agree}, e, c \rangle, \quad \mu_{\mathsf{pro}}(e, u) = \langle \mathsf{sid}, \mathsf{tag}_{\mathsf{PCSBB}}, \texttt{proposal}, e, u \rangle,$$

$$\mu_{\mathsf{vote}}(e, u) = \langle \mathsf{sid}, \mathsf{tag}_{\mathsf{PCSBB}}, \texttt{vote}, e, u \rangle.$$

Each MVBA instance and its coin calls use distinct subroutine identifiers derived from (sid, e) and separate signing domains. All steps have the stated round deadlines; missing or invalid messages are ignored.

Terminology. We follow standard SMR terminology.

- ***Epoch.*** Time is divided into synchronous epochs. In each epoch $e \geq 1$, one sender consortium serves as the leader consortium in round-robin order.
- ***Block, accumulator, certificate, and rank.*** In epoch e, the leader consortium forms a block $B_e = (\mathsf{sid}, e, \mathsf{msg}, \mathsf{cert}_{e'})$, where $\mathsf{cert}_{e'}$ is a certificate from an earlier epoch. Its accumulator is computed deterministically as

$$u = \mathsf{Acc.Eval}(ak, \{(j, c_j)\}_{j \in [n]}), \qquad (c_1, \ldots, c_n) \leftarrow \mathsf{EC.Encode}(B_e). \tag{2}$$

Since $\mathsf{EC.Encode}$ is injective and the accumulator is collision resistant, u serves as a digest of B_e. We write

$$(B_e, u) \leftarrow \mathsf{buildBlock}(\mathsf{sid}, e, \mathsf{cert}', \mathsf{msg}). \tag{3}$$

A certificate for B_e is $\mathsf{cert}_e = (\mathsf{sid}, e, u, \sigma)$, where

$$\mathsf{MTS.Vrfy}\big(\mathsf{VK}_{\mathcal{P}}, \sigma, \mu_{\mathsf{vote}}(e, u)\big) = 1. \tag{4}$$

Its *rank* is its epoch number e; blocks inherit the rank of their epoch. The genesis certificate $\mathsf{cert}_0 = (\mathsf{sid}, 0, \perp, \perp)$ is valid by convention.
A well-formed block has the current session and epoch, an admissible payload, and a valid parent certificate of smaller rank. A block directly extends its parent block and extends all of its parent's ancestors.
Each party maintains $\mathsf{CERTs}[e]$, containing the certificate for epoch e or $\perp$, and $\mathsf{BLKs}[e]$, containing a received valid block together with its commitment status, or $\perp$.

Main PCSBB protocol. As illustrated in Fig. 4, the leader consortium rotates every epoch. During the first m epochs, each consortium sequentially proposes its input message msg_i, while in the last m epochs, they propose only empty messages. The protocol concludes after $2m$ epochs. Its output vector is indexed by the first m epochs: slot e contains the message in the committed block created in epoch e, or $\perp$ if no such block is committed.

Each epoch consists of a joint execution denoted as PCSBB.Epoch, which follows a structured five-step process letting the leader consortium form a new block and propose the block for agreement. The details of these steps are depicted in Fig. 5 and Fig. 5, and described in text below. Step 2 invokes the intra-consortium consensus LeaderAgreeCert, presented in § 4.3.

PCSBB[sid]. Each P_i performs the following steps:

$\mathsf{CERTs}[0] \leftarrow \mathsf{cert}_0$, $\mathsf{BLKs} \leftarrow \emptyset$; $\mathsf{STATE} \leftarrow (\mathsf{CERTs}, \mathsf{BLKs})$
for $e \in [1, 2m]$ **do** // choose the leader consortium for epoch e
  $\mathcal{L}_e \leftarrow \mathcal{D}^{(e)}$ **if** $e \leq m$, otherwise $\mathcal{L}_e \leftarrow \mathcal{D}^{(e-m)}$
  if $i \in \mathcal{L}_e \wedge e \leq m$ **then** $\mathsf{msg} = \mathsf{msg}_i$ **else** $\mathsf{msg} = \perp$ // propose $\perp$ in last m epochs
  $\mathsf{STATE} \leftarrow \mathsf{PCSBB.Epoch}(\mathsf{sid}, e, \mathcal{P}, \mathcal{L}_e, \mathsf{msg}, \mathsf{STATE})$
// Epoch subroutine: Figs. 5 and 5
for $e \in [m]$ **do**
  if $\mathsf{BLKs}[e].\mathtt{committed}$ **then** $v_e \leftarrow \mathsf{BLKs}[e].B_e.\mathsf{msg}$ **else** $v_e \leftarrow \perp$
output $\mathcal{V} = (v_1, \ldots, v_m)$

Fig. 4. The PCSBB construction. Each epoch lasts for $\Delta_{\mathsf{epoch}} = 9 + \Delta_{\mathsf{MVBA}}$ time and each MVBA lasts for $\Delta_{\mathsf{MVBA}} = O(\kappa)$ time.

Epoch details. We describe PCSBB.Epoch in steps.

Step 1: Epoch initialization. Before the first epoch, every node initializes $\mathsf{CERTs}[0] = \mathsf{cert}_0$ and the distinguished genesis block B_0. In later epochs, nodes process certificates shared in the preceding epoch and update their highest-ranked block certificate accordingly. Each node also prepares to join the two DD instances used to disperse long messages in the current epoch. At the end of this step, each party sends its highest-ranked valid certificate to every node in the leader consortium.

Step 2: Propose. Only nodes in the leader consortium execute this step. Each leader node receives certificates and updates its knowledge of the highest-ranked certificate. This ensures that honest nodes within the consortium acknowledge the highest-ranked certificate known to any honest party in the network.

Within the consortium, nodes invoke the intra-consortium consensus protocol LeaderAgreeCert, which ensures agreement on a valid block certificate ranked at least as high as every certificate known to an honest node before the epoch. Next, nodes construct the new block B_e using the agreed highest certificate. The block is deterministically encoded into n fragments via EC.Encode, and its accumulator $u = \mathsf{DD.Com}(B_e)$ is computed. Each node then partially signs the domain-separated proposal tuple specified in Step 2, producing a signature π_i, and proposes the new block by multicasting a pro-short message containing u and π_i.

Step 1 (Epoch initialization): Timer $T \leftarrow 0$ //increasing by 1 every sync. round.

1 : $\mathsf{hcert}^* \leftarrow$ highest-ranked certificate in CERTs
2 : **if** $e > 1$
3 : receive$\{(\mathsf{sid}, \mathsf{new\text{-}cert}, e-1, \mathsf{hcert}_j)\}_{j \in H_1}$ // messages from epoch $e-1$
4 : $\mathsf{hcert}^* \leftarrow$ highest-ranked valid certificate in $\{\mathsf{hcert}^*\} \cup \{\mathsf{hcert}_j\}_{j \in H_1}$
5 : Add hcert^* to CERTs
6 : **send** $(\mathsf{sid}, \mathsf{highest\text{-}cert}, e, \mathsf{hcert}^*)$ to all current leaders $P_j \in \mathcal{L}_e$

Step 2 (Propose): start at $T = 1$

1 : **if** $P_i \in \mathcal{L}_e$ **then** // if P_i is in the leader consortium
2 : receive $\{(\mathsf{sid}, \mathsf{highest\text{-}cert}, e, \mathrm{hcert}_j)\}_{j \in H_2}$
3 : $\mathsf{hcert}_i^* \leftarrow$ highest-ranked valid certificate in $\{\mathsf{hcert}^*\} \cup \{\mathsf{hcert}_j\}_{j \in H_2}$
4 : // agree on the highest certificate within consortium $\mathcal{L}_e$
5 : $\mathsf{hcert}_{\mathsf{agreed}} \leftarrow \mathsf{LeaderAgreeCert}\langle \mathsf{sid}, e, \mathcal{L}_e, P_i(\mathsf{hcert}_i^*)\rangle$
6 : // build B_e that extends $\mathsf{hcert}_{\mathsf{agreed}}$ and includes msg with accumulator u
7 : $(B_e, u) \leftarrow \mathsf{buildBlock}(\mathsf{sid}, e, \mathsf{hcert}_{\mathsf{agreed}}, \mathsf{msg})$ (cf. Eq. 3)
8 : $\pi_i \leftarrow \mathsf{MTS.PartSign}(msk_i, \mu_{\mathsf{pro}}(e, u))$
9 : // use DD to disseminate B_e, directly multicast the signed accumulator u
10 : $\mathsf{DD.send}[\mathsf{sid}, e, \mathsf{pro\text{-}long}]\,(B_e, u)$, **multicast** $(\mathsf{sid}, \mathsf{pro\text{-}short}, e, u, \pi_i)$

Fig. 5. The PCSBB epoch construction PCSBB.Epoch: Step 1-2

Remarkably, our LeaderAgreeCert is based on MVBA, which terminates within $\Delta_{\mathsf{MVBA}} + 3$ rounds. This step is where we leverage endemic coins.

Step 3: Vote and redeliver. At time $T = 5 + \Delta_{\mathsf{MVBA}}$, if a node receives $\lfloor \frac{|\mathcal{L}_e|}{2} \rfloor + 1$ pro-short messages that agree on the same accumulated value u^* and provide valid partial signatures, the node will accept the short proposal u^*, aggregate $\lfloor \frac{|\mathcal{L}_e|}{2} \rfloor + 1$ valid partial signatures into a threshold signature π on that proposal tuple, and announce its approval on u^* to others by multicasting a pro-accept message containing the leader-proposal certificate $(\mathsf{sid}, e, u^*, \pi)$. When a node accepts u^*, it does two more things before voting for u^*. First, it calls $\mathsf{DD.relay}[\mathsf{sid}, e, \mathsf{pro\text{-}long}](u^*)$ to relay its indexed fragment. Second, at $T = 6 + \Delta_{\mathsf{MVBA}}$, it calls $\mathsf{DD.recover}[\mathsf{sid}, e, \mathsf{pro\text{-}long}](u^*)$ and proceeds only if the output is a non-$\perp$ well-formed block, in which case it updates its highest-ranked certificate accordingly. Note that it is important to update the highest certificate; otherwise the current node may refuse to vote a valid proposal since the proposal may not extend the highest-ranked certificate prior to the update. Later at $T = 7 + \Delta_{\mathsf{MVBA}}$, if no equivocation has been detected in Step 4, it invokes $\mathsf{DD.send}[\mathsf{sid}, e, \mathsf{redeliver}]$ to ensure availability of long proposal among all honest nodes, and votes for u^* by multicasting a partial signature ρ_i on the domain-separated vote tuple specified in Step 3.

Step 4: Accept. At time $T = 6 + \Delta_{\mathsf{MVBA}}$, nodes receive pro-accept messages and validate each corresponding proof π_j^* on the proposal tuple specified in Step 4. When conflicting acceptance of two different blocks is identified, a node alerts others by multicasting the equivocating leader consortium threshold signatures on the two proposal tuples.

Step 5: Commit. At $T = 8 + \Delta_{\mathsf{MVBA}}$, each node with an accepted accumulator $u^* \neq \perp$ calls $\mathsf{DD.relay}[\mathsf{sid}, e, \mathsf{redeliver}](u^*)$ to relay its indexed fragment for the accepted accumulator. When a node receives $\lfloor \frac{n}{2} \rfloor + 1$ valid votes on the vote tuple specified in Step 5, it aggregates these votes into a block certificate $(\mathsf{sid}, e, u^*, \sigma)$ and announces the new certificate to others. At $T = 9 + \Delta_{\mathsf{MVBA}}$, every node processes valid new-cert messages and invokes $\mathsf{DD.recover}[\mathsf{sid}, e, \mathsf{redeliver}](u^*)$ if it does not already store the block. A node

Step 3 (Vote and redeliver): start at $T = 5 + \Delta_{\mathsf{MVBA}}$

initialize $u^* \leftarrow \perp$ // u^* stores the accumulator of the new block
receive $\{(\mathsf{sid}, \text{pro-short}, e, u_j, \pi_j)\}_{j \in H_3}$ // signed accumulators
if $\exists u'$ s.t. there are at least $\lfloor \frac{|\mathcal{L}_e|}{2} \rfloor + 1$ valid $\{u', \pi_j\}_{j \in H_3}$
// if $> 50\%$ leaders signed the same u_j, decide $u^* = u_j$ and generate thld.sig.
  $u^* \leftarrow u'$, aggregate the partial signatures into a threshold signature π
  //multicast u^* with the thld.sig. π; relay fragments consistent with u^*
  multicast $(\mathsf{sid}, \text{pro-accept}, e, u^*, \pi)$; DD.relay[sid, e, pro-long](u^*)
  // $T = 6 + \Delta_{\mathsf{MVBA}}$, recover B_e with fragments consistent with u^*
  $B_e \leftarrow$ DD.recover[sid, e, pro-long] (u^*)
  if $B_e \neq \perp$ **then** parse $B_e \rightarrow (\mathsf{sid}, e, \mathsf{msg}, \mathsf{cert}')$
    BLKs[e] $\leftarrow (B_e, \texttt{false})$// store the uncommitted block
    if cert$'$ is valid and $\geq_{\mathsf{rank}}$ hcert* **then** Add cert$'$ to CERTs
  // $T = 7 + \Delta_{\mathsf{MVBA}}$:
  if B_e is well-formed $\wedge$ cert$'$ is the highest in CERTs $\wedge$ No Equivocation **then**
    DD.send[sid, e, redeliver] (B_e, u^*) // re-disseminate the new block
    $\rho_i \leftarrow$ MTS.PartSign$(msk_i, \mu_{\mathsf{vote}}(e, u^*))$; **multicast** $(\mathsf{sid}, \mathsf{vote}, e, u^*, \rho_i)$

Step 4 (Accept): start at $T = 6 + \Delta_{\mathsf{MVBA}}$

receive $\{(\mathsf{sid}, \text{pro-accept}, e, u_j^*, \pi_j^*)\}_{j \in H_4}$
for $j \in H_4$ **do** **if** MTS.Vrfy$(\mathsf{VK}_{\mathcal{L}_e}, \pi_j^*, \mu_{\mathsf{pro}}(e, u_j^*)) = 1$ **then**
    if $u^* = \perp$ **then** $(u^*, \pi) \leftarrow (u_j^*, \pi_j^*)$
    else if $u_j^* \neq u^*$ **then multicast** $(\mathsf{sid}, \mathsf{equivocate}, e, u^*, \pi, u_j^*, \pi_j^*)$

Step 5 (Commit): start at $T = 8 + \Delta_{\mathsf{MVBA}}$

if $u^* \neq \perp$ **then** DD.relay[sid, e, redeliver](u^*)// help relay B_e.
receive $\{(\mathsf{sid}, \mathsf{vote}, e, u_j^*, \rho_j)\}_{j \in H_5}$
if At least $\lfloor \frac{n}{2} \rfloor + 1$ valid $\{u^*, \rho_j\}_{j \in H_5}$
// if $> 50\%$ nodes voted for (e, u^*), a certificate for B_e can be generated
   aggregate the partial signatures into a threshold signature σ
  CERTs[e] $\leftarrow (\mathsf{sid}, e, u^*, \sigma)$, **multicast** $(\mathsf{sid}, \text{new-cert}, e, \mathsf{CERTs}[e])$
// $T = 9 + \Delta_{\mathsf{MVBA}}$:
if BLKs[e] $= \perp$ **then** $B_e \leftarrow$ DD.recover[sid, e, redeliver](u^*); BLKs[e] $\leftarrow (B_e, \texttt{false})$
receive $\{(\mathsf{sid}, \text{new-cert}, e, \mathsf{cert}_j)\}_{j \in H_6}$
if $\exists$ a valid cert$_j$ for (sid, e), **then** CERTs[e] $\leftarrow$ cert$_j$
if have sent $(\mathsf{sid}, \text{new-cert}, e, \dots) \wedge B_e \neq \perp \wedge$ No Equivocation **then**
  Update BLKs[e'].committed $\leftarrow$ true for any e' s.t. $e' = e$ or B_e extends $B_{e'}$
STATE$'$ $\leftarrow$ (CERTs, BLKs), **output** STATE$'$

Fig. 5. The PCSBB epoch: Steps 3–5 (Continued).

directly commits only if it locally formed and announced the certificate in the preceding round and has a non-$\perp$ recovered block; a certificate received from another party is stored for use in subsequent epochs but does not by itself trigger direct commitment. Thus, whenever one honest party directly commits, its certificate has already been multicast for a full round. Lemma 5 ensures that whenever a valid block certificate is formed in an epoch, all honest parties recover the same block by this deadline.

Step Equivocation. If an honest party detects equivocation in Step 4, it multicasts the proof of equivocation at time $T = 6 + \Delta_{\mathsf{MVBA}}$. Hence at time $T = 7 + \Delta_{\mathsf{MVBA}}$ honest parties receive the proof and abstain from voting and direct commitment in epoch e. They still complete Step 5's data relay and recovery and retain valid certificates. See Fig. 6 for the detailed actions.

We present the five steps of PCSBB.Epoch in Fig. 5, Fig. 5. We describe the details of the equivocation step in Fig. 6.

Step Equivocation: $T = 7 + \Delta_{\mathsf{MVBA}}$

1 : **receive** $(\mathsf{sid}, \mathsf{equivocate}, e, u[1], \pi[1], u[2], \pi[2])$ **then**
2 : **if** $u[1] \neq u[2] \wedge \mathsf{MTS.Vrfy}(\mathsf{VK}_{\mathcal{L}_e}, \pi[1], \mu_{\mathsf{pro}}(e, u[1])) = 1 \wedge \mathsf{MTS.Vrfy}(\mathsf{VK}_{\mathcal{L}_e}, \pi[2], \mu_{\mathsf{pro}}(e, u[2])) = 1$ **then**
3 : $\mathsf{abstain} \leftarrow \mathtt{true}$ // stop voting and directly committing
4 : Continue Step 5 to relay, recover, and retain certified blocks.

Fig. 6. The PCSBB Epoch construction: Step Equivocation

Remark 1. An equivocation proof received after $T = 7 + \Delta_{\mathsf{MVBA}}$ is ignored for the current epoch. A proof delivered selectively at the deadline can make only some honest parties abstain, which is why abstention must not stop data relay, recovery, or certificate storage.

4.3 The Intra-Consortium Consensus LeaderAgreeCert

In PCSBB, a critical step is to ensure that the leader consortium agrees on a certificate satisfying the required rank lower bound, so that they can jointly act as a leader to propose a new block and help the network commit previous blocks. However, in Step 1, malicious nodes may send different certificates with different ranks to different consortium members, creating inconsistencies in their views. We introduce an intra-consortium consensus LeaderAgreeCert, used in Step 2 of PCSBB.Epoch, to solve this issue. In LeaderAgreeCert, each $P_i \in \mathcal{D}$ provides the highest-ranked valid certificate cert_i known to it and agrees on a valid certificate cert whose rank is at least the minimum honest input rank. After Step 1, this includes every certificate reported by an honest party before that exchange.

Its pseudo-code is given below.

LeaderAgreeCert$\langle \mathsf{sid}, e, \mathcal{D}, P_i(\mathrm{cert}_i)\rangle$

1 : **send** $(\mathsf{sid}, e, \mathsf{request}, \mathsf{cert}_i)$ to all $P_j \in \mathcal{D}$// propose cert_i
2 : **receive** $\{(\mathsf{sid}, e, \mathsf{request}, \mathsf{cert}_j)\}_{j\in C_1}$ // receive candidate certificates
3 : **for** $j \in C_1$ **do** **if** cert_j is valid and $\mathsf{cert}_j \geq_{\mathsf{rank}} \mathsf{cert}_i$ **then**
4 : $\gamma_{i,j} \leftarrow \mathsf{MTS.PartSign}(msk_i, \mu_{\mathsf{lac}}(e, \mathsf{cert}_j))$, **send** $(\mathsf{sid}, e, \mathsf{signed}, \gamma_{i,j})$ to P_j
5 : **receive** $\{(\mathsf{sid}, e, \mathsf{signed}, \gamma_{j,i})\}_{j\in C_2}$ // receive partial signatures on cert_i
6 : **if** there are $\geq \lfloor|\mathcal{D}|/2\rfloor + 1$ valid partial signatures on cert_i **then**
7 : Aggregate them into a threshold signature α_i
8 : **send** $(\mathsf{sid}, e, \mathsf{tsig}, \alpha_i, \mathsf{cert}_i)$ to all $P_j \in \mathcal{D}$// multicast the thld.sig. α_i
9 : **receive** $\mathsf{TSIGs} = \{(\mathsf{sid}, e, \mathsf{tsig}, \alpha_j, \mathsf{cert2}_j)\}_{j\in C_3}$
10 : **pick** $(\alpha, \mathsf{cert2})$ from TSIGs s.t. cert2 has the highest rank and $Q(\alpha, \mathsf{cert2}) = 1$
11 : // $Q(\alpha, c) := c$ is valid $\wedge$ $\mathsf{MTS.Vrfy}(\mathsf{VK}_{\mathcal{D}}, \alpha, \mu_{\mathsf{lac}}(e, c)) = 1$.
12 : $(\beta, \mathsf{cert3}) \leftarrow \mathsf{MVBA}\langle P_i\,((\alpha, \mathsf{cert2}))\,, \mathsf{predicate} = Q\rangle$, **return** cert3

Protocol Description. Let $\eta = |\mathcal{D}|$ and $q_{\mathcal{D}} = \lfloor \eta/2 \rfloor + 1$. Each member $P_i \in \mathcal{D}$ starts with its highest-ranked valid certificate cert_i. The protocol first uses three communication rounds to prepare inputs for MVBA. First, every member multicasts its certificate within $\mathcal{D}$. Second, a member endorses each received valid certificate whose rank is at least its own input rank, returning a partial signature on the leader-agreement tuple. Third, a proposer that collects $q_{\mathcal{D}}$ valid endorsements combines them into a threshold signature α and multicasts (α, cert) within $\mathcal{D}$.

Each member selects a received pair satisfying the public predicate Q, preferring the highest rank. This predicate checks that c is a valid certificate and that α is a valid consortium threshold signature endorsing c in this session and epoch. Members invoke MVBA on their selected pairs with external predicate Q, and return the certificate in the agreed pair. The endemic coin of $\mathcal{D}$ supplies the randomness required by MVBA.

Security Analysis. Assume that $\mathcal{D}$ has an honest majority and that the underlying MTS and MVBA are secure. The following properties hold except with negligible probability.

Agreement. All honest members return the same certificate by MVBA agreement.

Validity. The output is a valid certificate whose rank is at least that of every certificate known to all honest members at the start of the subroutine. To see this, let C be any such certificate. Every honest input has rank at least $\mathsf{rank}(C)$. By MVBA external validity, the output pair (α, c) satisfies Q. Since α requires a majority of endorsements, MTS unforgeability implies that at least one honest member endorsed c. Its signing rule gives

$$\mathsf{rank}(c) \geq \mathsf{rank}(\mathsf{cert}_i) \geq \mathsf{rank}(C).$$

In PCSBB, Step 1 sends every honest party's highest certificate to all consortium members. Thus the returned certificate ranks at least as high as every certificate known to an honest party before the epoch begins.

Termination. Let $\mathsf{cert}_{\max}$ be a highest-ranked input of an honest member. Every honest member endorses it, so its proposer collects at least $q_{\mathcal{D}}$ endorsements and multicasts a pair satisfying Q. Consequently, every honest member has a valid MVBA input. MVBA termination then gives termination of LeaderAgreeCert; with the prescribed timeout, it completes within $3 + \Delta_{\mathsf{MVBA}}$ rounds except with the failure probability in Lemma 2.

Efficiency Analysis. A block certificate has $O(\lambda \log n)$ bits, and an endorsement proof has $O(\lambda \log \eta)$ bits. The three preparatory rounds send $O(\lambda \eta^2 \log n)$ bits in total. MVBA agrees on a pair of $O(\lambda \log n)$ bits. We stop the instance after $O(\kappa)$ rounds, bounding its communication cost by

$$O\big(\kappa(\eta^2 \lambda \log \eta + \eta \lambda \log n)\big)$$

bits, including when the consortium has a corrupt majority. Adding the preparatory rounds gives a total communication bound of

$$O\big(\lambda\eta^2 \log n + \kappa(\eta^2\lambda \log \eta + \eta\lambda \log n)\big)$$

bits for LeaderAgreeCert, within a round budget of $3 + \Delta_{\mathsf{MVBA}} = O(\kappa)$.

4.4 The Data Delivery Subroutine

Let $k := \lfloor n/2 \rfloor + 1$. The DD subroutine uses the fixed deterministic (k, n) erasure code from Section 4.2. For a message M, define

$$\mathsf{DD.Com}(M) := \mathsf{Acc.Eval}(ak, \{(j, c_j)\}_{j\in[n]}), \qquad (c_1, \ldots, c_n) := \mathsf{EC.Encode}(M).$$

We use accumulator unforgeability as defined in § 3.3: except with negligible probability, no PPT adversary can produce a valid membership witness for an element outside the set accumulated by u. For our Merkle-tree instantiation, this property follows from collision resistance of the underlying hash function. Each execution is identified by a unique did, and authenticated channels bind every received message to its sender. In PCSBB, $\mathsf{did} := \langle \mathsf{sid}, e, \mathsf{label}\rangle$; for readability, we write this identifier as $[\mathsf{sid}, e, \mathsf{label}]$ in the algorithms. The subroutine is described in Fig. 7.

Lemma 3 (DD availability and integrity).

- *Availability: Fix an instance* did *and a message* M*. Let* $u = \mathsf{DD.Com}(M)$*. Suppose one honest party invokes* DD.send *with* (M, u) *in the dispersal round, and every honest party subsequently invokes* DD.relay *with* u *in the relay round. All these calls use the same instance* did*. Then, except with negligible probability, every honest party can recover* M *by the end of the relay round: a subsequent call to* DD.recover *with that instance and* u *returns* M*.*
- *Integrity: If two honest parties invoke* DD.recover *with the same* did *and* u *and output* M *and* M'*, respectively, then* $\Pr[M = M'] \geq 1 - \mathsf{negl}(\lambda)$*.*

Proof. For availability, let

$$\mathcal{S} := \{(j, c_j)\}_{j\in[n]}, \qquad (c_1, \ldots, c_n) := \mathsf{EC.Encode}(M),$$

so that $u = \mathsf{Acc.Eval}(ak, \mathcal{S})$. An honest sender supplies every party P_j with a valid witness for (j, c_j). Suppose a malicious sender could instead supply a valid tuple (j, c'_j, w'_j) under u with $c'_j \neq c_j$. Because $\mathcal{S}$ contains exactly one indexed fragment at position j, we have $(j, c'_j) \notin \mathcal{S}$. Such a tuple would constitute a forgery against accumulator unforgeability. Consequently, except with negligible probability, every valid tuple accepted by P_j under u contains the correct fragment c_j. Since P_j receives such a tuple from the honest sender, it multicasts (j, c_j, w_j) when invoking $\mathsf{DD.relay}[\mathsf{did}](u)$.

There are at least $n - t \geq \lfloor n/2 \rfloor + 1 = k$ honest parties. Thus every honest recovering party receives at least k correct fragments at distinct positions from their designated relayers. These fragments decode to M, and the final re-encoding and accumulator check succeeds, so every honest party outputs M.

For integrity, suppose two honest recovery calls under the same (did, u) output distinct messages M and M'. Let $\mathcal{S}$ and $\mathcal{S}'$ be their respective sets of indexed encoded fragments. The final checks in DD.recover imply

$$\mathsf{Acc.Eval}(ak, \mathcal{S}) = u = \mathsf{Acc.Eval}(ak, \mathcal{S}').$$

Because the erasure code is injective, $M \neq M'$ implies $\mathcal{S} \neq \mathcal{S}'$. Hence there is an element $s' \in \mathcal{S}' \setminus \mathcal{S}$. Accumulator correctness gives a valid membership witness for s' under $\mathsf{Acc.Eval}(ak, \mathcal{S}') = u$; the same witness verifies against $\mathsf{Acc.Eval}(ak, \mathcal{S}) = u$, even though $s' \notin \mathcal{S}$. This contradicts accumulator unforgeability. Therefore $M = M'$ except with negligible probability. □

For completeness, at most n parties invoke DD.send in one instance. Each sends n fragments of size $O(\ell/n)$ with $O(\lambda \log n)$-bit witnesses, for a total of $O(n\ell + \lambda n^2 \log n)$ bits. In the relay round, at most n honest parties multicast one fragment and witness, which has the same asymptotic cost.

DD.send[did](msg, u^*). Calling party P_s performs:

// Deterministically encode and disperse one fragment per recipient.
$(c_1, \ldots, c_n) \leftarrow$ EC.Encode(msg)
$u \leftarrow$ Acc.Eval$(ak, \{(j, c_j)\}_{j\in[n]})$; **if** $u \neq u^*$ **then return** $\perp$
for $j \in [n]$ **do**
  $w_j \leftarrow$ Acc.Wit$(ak, u^*, (j, c_j))$
  send $(\mathsf{dd}, \mathsf{did}, \mathsf{disperse}, u^*, j, c_j, w_j)$ to P_j

DD.relay[did](u^*). Calling party P_i performs:

// Multicast at most one valid opening for position i.
receive $\{(\mathsf{dd}, \mathsf{did}, \mathsf{disperse}, u_s, i, c_{s,i}, w_{s,i})\}_{s\in S_i}$
for $s \in S_i$ **do**
  if $u_s = u^* \wedge$ Acc.Vrfy$(ak, u^*, (i, c_{s,i}), w_{s,i}) = 1$ **then**
    multicast $(\mathsf{dd}, \mathsf{did}, \mathsf{relay}, u^*, i, c_{s,i}, w_{s,i})$; **return**

DD.recover[did](u^*). Calling party P_i performs:

// Accept indexed fragments only from their designated relayers.
$C \leftarrow \emptyset$; **receive** $\{(\mathsf{dd}, \mathsf{did}, \mathsf{relay}, u_j, j, c_j, w_j)\}_{j\in R_i}$
for $j \in R_i$ **do**
  if $u_j = u^* \wedge$ Acc.Vrfy$(ak, u^*, (j, c_j), w_j) = 1$ **then** $C[j] \leftarrow c_j$
if $|C| \geq k$ **then**
  msg $\leftarrow$ EC.Decode(C); $(\widehat{c}_1, \ldots, \widehat{c}_n) \leftarrow$ EC.Encode(msg)
  $\widehat{u} \leftarrow$ Acc.Eval$(ak, \{(j, \widehat{c}_j)\}_{j\in[n]})$
  if $\widehat{u} = u^*$ **then return** msg
return $\perp$

Fig. 7. The Data Delivery protocol, where $k = \lfloor n/2 \rfloor + 1$ and messages are received over authenticated channels.

4.5 The Analysis

Efficiency Analysis. For round complexity, each epoch takes $9 + \Delta_{\mathsf{MVBA}}$ rounds. As a PCSBB instance consists of $2m$ epochs, it takes $2m(9 + \Delta_{\mathsf{MVBA}})$ rounds in total. Under the global condition $\log N = O(\kappa)$, Lemma 2 permits $\Delta_{\mathsf{MVBA}} = O(\kappa)$, giving $O(\kappa m)$ rounds. In general the deadline is $O(m(\kappa + \log N))$, with the same replacement in the cutoff-dependent communication term below.

For communication complexity, let η be the size of the leader consortium in the current epoch. In Step 1, each party sends an $O(\lambda \log n)$-bit certificate to every leader, costing $O(\lambda n \eta \log n)$ bits.

In Step 2, the preparatory rounds of $\mathsf{LeaderAgreeCert}$ cost $O(\lambda \eta^2 \log n)$ bits, and its time-limited MVBA instance costs at most $O\big(\kappa(\eta^2 \lambda \log \eta + \eta \lambda \log n)\big)$ bits. The leaders also multicast short proposals and use DD to disperse blocks of size $O(\ell + \lambda \log n)$. These actions cost $O(\eta n \lambda + n\ell + \lambda n^2 \log n)$ bits.

Steps 3–5 use one more DD instance and a constant number of $O(\lambda \log n)$-bit multicasts per honest party. Each honest party also multicasts at most one equivocation proof. Their total cost is $O(n\ell + \lambda n^2 \log n)$ bits. Since $\eta \le n$, the communication of one epoch is bounded by

$$O\big(n\ell + \lambda n^2 \log n + \kappa(\eta^2 \lambda \log \eta + \eta \lambda \log n)\big).$$

Taking $\eta := \max_{j \in [m]} |\mathcal{D}^{(j)}|$, all $2m$ epochs together cost at most

$$O\big(mn\ell + \lambda m n^2 \log n + \kappa m(\eta^2 \lambda \log \eta + \eta \lambda \log n)\big) \tag{5}$$

bits. This bound holds also in epochs with a corrupt-majority leader consortium. These are bounded communication costs, without an expectation over MVBA's running time.

Security Analysis. We establish the security in the following theorem.

Theorem 1 (PCSBB Security). *Let* $\mathsf{PCSBB}[\mathsf{sid}]$ *be an instance operating under the transparent public-parameter setup of § 3 and an established intra-consortium one-round common coin. Under the SXDH assumption and the security of the intra-consortium common coin,* $\mathsf{PCSBB}[\mathsf{sid}]$ *guarantees termination, agreement, and validity in the random oracle model. Moreover,* $\mathsf{PCSBB}[\mathsf{sid}]$ *remains secure even if the adversary queries the signing oracle of the MTS scheme for messages outside this PCSBB session domain, namely canonical encodings whose first two fields are not* $(\mathsf{sid}, \mathsf{tag}_{\mathsf{PCSBB}})$.

Proof of Theorem 1. At a high level, we establish the following facts: (1) In every epoch at most one block can be certified (Lemma 4); (2) Once a certificate is generated in epoch e, every honest node receives the unique corresponding block B_e in that epoch (Lemma 5); (3) Every committed block must be certified (Lemma 6); (4) A committed block at a later epoch must extend a committed block of an earlier epoch (Lemma 7); (5) Under an honest-majority leader consortium, all honest nodes can commit the proposed B_e by the end of that epoch (Lemma 8).

When at least one sender consortium has an honest majority and the consortiums take turns as leader, all honest nodes commit the same blocks during the first m epochs, ensuring agreement. Validity follows from (5). □

We now prove these facts in turn.

We follow five steps: certificate uniqueness, block availability, certification of committed blocks, consistency across epochs, and commitment under an honest leader consortium. Throughout, certificates and signatures are verified on their prescribed session- and epoch-specific tuples. A well-formed block B_e contains a valid parent certificate of rank strictly less than e. Highest certificates are retained monotonically, and certificates of equal rank are compared by the block they identify, rather than by their signature encodings. A block is committed *directly* when Step 5 commits the proposal of the current epoch, and *indirectly* when it is an ancestor of a directly committed block.

We use the following equivocation rule: a party that detects equivocation stops voting and directly committing in that epoch, but still completes the scheduled data relay and recovery and stores valid block

certificates. This distinction is needed when an adversary selectively releases an equivocation proof at the voting deadline. Proofs received after that deadline do not affect the epoch.

Condition on MTS unforgeability, accumulator integrity, and the security of the underlying MVBA instances with honest-majority participants. The cryptographic failure probability is negligible. In the adaptive setting, "honest" refers to parties that remain uncorrupted throughout the execution. Since $q := \lfloor n/2 \rfloor + 1 > t$, every valid block certificate requires a vote from such a party.

Lemma 4 (Certificate uniqueness). *At most one block can be certified in each epoch.*

Proof. Suppose distinct blocks B and B' are certified in epoch e, with accumulators u and u'. Each certificate requires an honest vote, and an honest voter first recovers its block. Accumulator integrity implies $u \neq u'$. Before voting, the two honest voters multicast valid pro-accept messages for u and u' at $T = 5 + \Delta_{\mathsf{MVBA}}$. Both messages reach every honest party by $T = 6 + \Delta_{\mathsf{MVBA}}$. Hence the voters detect equivocation and abstain at $T = 7 + \Delta_{\mathsf{MVBA}}$, a contradiction. □

Lemma 5 (Block uniqueness and availability). *If a valid certificate for B_e exists, then every honest party stores the same block B_e in $\mathsf{BLKs}[e]$ by the end of epoch e, regardless of when the certificate is disclosed.*

Proof. A valid certificate contains an honest vote. Let P be such a voter, and let u be the certified accumulator. Before voting, P recovers B_e and multicasts a valid pro-accept message for u. Therefore every honest party receives u by $T = 6 + \Delta_{\mathsf{MVBA}}$. If an honest party had accepted or received a valid acceptance for a different accumulator by then, it would multicast an equivocation proof, reaching P before voting. Thus the existence of P's vote implies that every honest party finishes Step 4 with $u^* = u$.

At $T = 7 + \Delta_{\mathsf{MVBA}}$, P invokes DD.send for (B_e, u) in the redeliver instance. All honest parties relay for u at $T = 8 + \Delta_{\mathsf{MVBA}}$, including any party that subsequently received an equivocation proof and abstained from voting. Lemma 3 gives recovery of B_e by $T = 9 + \Delta_{\mathsf{MVBA}}$. Accumulator integrity identifies this recovered block with any copy already stored from pro-long, and Lemma 4 excludes a different certified block for the same epoch. □

Lemma 6 (Committed blocks are certified). *Every non-genesis block committed by an honest party is certified.*

Proof. Step 5 directly commits a block only after locally forming and announcing its certificate. Every indirectly committed block lies on the parent chain of a directly committed block. Each parent link contains a valid certificate for the parent block, so every block committed along this chain is also certified. □

Lemma 7 (Consistency across epochs). *If an honest party directly commits B_e in epoch e, then every certified block B_f with $f > e$ extends B_e. Consequently, no two blocks committed by honest parties conflict.*

Proof. Direct commitment requires announcing the certificate for B_e at $T = 8 + \Delta_{\mathsf{MVBA}}$. Thus every honest party receives it before epoch $e + 1$, and thereafter retains a highest certificate of rank at least e.

We prove the first claim by induction on $f > e$. A certificate for B_f requires an honest voter P. By the voting rule, the parent certificate of B_f identifies a highest-ranked certified block known to P. Its rank r therefore satisfies $e \leq r < f$. If $r = e$, certificate uniqueness identifies the parent with B_e. If $e < r < f$, the induction hypothesis says that the parent extends B_e. In either case, B_f extends B_e, proving the induction.

For the consequence, any two committed blocks lie on the ancestor chains of two directly committed blocks. If the latter have the same rank, they coincide by certificate uniqueness. Otherwise, the higher-ranked one extends the lower-ranked one by the first claim and Lemma 6. Their ancestor chains are therefore compatible. □

Lemma 8 (Commitment under an honest leader consortium). *Suppose the leader consortium in epoch e has an honest majority and all its honest members have the same input* msg*. If its MVBA instance finishes within the allotted time, every honest party directly commits the same block B_e containing* msg *by the end of the epoch, and commits all its ancestors.*

Proof. Let C be a highest-ranked certificate known to an honest party before epoch e. Step 1 delivers a certificate of rank at least $\mathsf{rank}(C)$ to every honest leader. The analysis in § 4.3 shows that LeaderAgreeCert returns the same valid parent certificate C' to all honest leaders, with $\mathsf{rank}(C') \geq \mathsf{rank}(C)$.

The common input and deterministic block construction give all honest leaders the same block B_e and accumulator u. Their majority supplies enough proposal signatures to every honest party, and no different accumulator can receive a valid proposal certificate: such a certificate would require an honest leader's signature. Hence no valid equivocation proof exists. All honest parties accept u and relay in pro-long, so DD availability lets them recover B_e by $T = 6 + \Delta_{\mathsf{MVBA}}$.

On recovery, each party adds C' to its certificate map if its rank is at least its previous highest rank. Certificate uniqueness handles ties, so every honest party regards C' as a highest certificate and votes for B_e. The $n - t \geq q$ honest votes allow every honest party to form and announce a certificate by $T = 8 + \Delta_{\mathsf{MVBA}}$, and directly commit B_e at $T = 9 + \Delta_{\mathsf{MVBA}}$. All ancestors are certified and were made available in their respective epochs by Lemma 5, so they are committed as well. □

PCSBB Security. We now obtain the three properties in Definition 2.

Lemma 9 (Agreement). *If an honest consortium exists and its second-cycle MVBA finishes within the allotted time, all honest parties output the same m-element vector.*

Proof. Let $\mathcal{D}^{(j)}$ be an honest consortium. In its second epoch $e^* = m + j$, all honest leaders have the common default input $\perp$. By Lemma 8, every honest party commits the same block B^* and all its ancestors.

Consider any block X of rank at most m that an honest party commits by termination, and let Y be a directly committed block whose ancestor chain contains X, possibly $Y = X$. If Y has rank below e^*, consistency across epochs implies that B^* extends Y; if their ranks are equal, certificate uniqueness gives $Y = B^*$. If Y has rank above e^*, it extends B^*. Since ranks strictly increase along a chain and $\mathsf{rank}(X) \leq m < e^*$, X must then precede B^* on that chain. Thus every committed first-cycle block lies on the common chain ending at B^*, which every honest party commits. Output extraction consequently gives the same message in every represented slot and $\perp$ in every other slot. □

Validity. If $\mathcal{D}^{(i)}$ is honest and its honest members share msg_i, Lemma 8 gives a common direct commitment of a block containing msg_i in epoch i. Hence every honest party outputs $v_i = \mathsf{msg}_i$.

Termination and failure probability. PCSBB advances after each fixed epoch deadline and outputs after exactly $2m$ epochs, even if a faulty consortium's MVBA does not finish. Its total duration is $2m(9 + \Delta_{\mathsf{MVBA}}) = O(\kappa m)$ rounds. The timeout choice of Lemma 2, applied only to instances with honest-majority participants, bounds the probability of any required MVBA timeout by $2^{-\kappa}$. Together with cryptographic security, agreement and validity therefore hold except with probability $\mathsf{negl}(\lambda) + 2^{-\kappa}$.

This completes the proof of Theorem 1. □

5 Near-Optimal DKG for One-Round Common Coin

In this section, we present a group-element DKG protocol with $\widetilde{O}(n^2)$ communication complexity and $\widetilde{O}(1)$ rounds, for the one-round common coin protocol [16] (which we recall in § 5.1). As a warm-up, we first present a simple protocol with sub-linear rounds in § 5.2, and then our recursive protocol RDKG in § 5.3.

5.1 Bacho et al.'s One-Round Common Coin

For completeness, we include a description of Bacho et al.'s common coin [16] in the following. Here we only specify the key structure required by the scheme, without giving all details of the setup and distributed key generation parts, as they will be fulfilled by our new group-element DKG protocols in this section.

Building Block. The scheme uses the same pairing groups as APVSS, with parameters $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g, h)$. It also uses a NIZK proof system, dubbed Dleq, for showing two group elements have the same discrete logarithm w.r.t. two different bases. Formally, Dleq works for the following relation:

$$R_{eq} := \{(g_1, g', g_2, g'') \in \mathbb{G}_1^4; \alpha : g' = g_1^\alpha \wedge g'' = g_2^\alpha\}.$$

Note that Dleq has an efficient instantiation in the random oracle model [31]. In addition, it assumes a random oracle $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and a random oracle $H_2 : \mathbb{G}_T \to V$.

Description. The description of the protocol is available in Fig. 8.

Setup. The setup phase consists of a DKG setup and a one-round *commitment* setup.

- *DKG:* Each participant P_i obtains $(pk, pk_1, \ldots, pk_n) \in \mathbb{G}_1^{n+1}$ and its secret key $sk_i \in \mathbb{G}_2$. The output satisfies that $e(pk_i, h) = e(g, sk_i)$, and there exists a t-degree polynomial f, s.t. $pk_j = g^{f(j)}$ for $j \in [n]$ and $pk = g^{f(0)}$.
- $\mathsf{ComSetup}$: In addition to the DKG setup, each honest participant P_i performs the following tasks in one round:
  - Initialize a local set $\mathcal{G} = \emptyset$.
  - Sample $\alpha_i \leftarrow\!\!\$\ \mathbb{Z}_p^*$, and multicast $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} sk_i)$. Store α_i.
  - Upon receiving $\mathsf{cm}_j = (\mathsf{cm}_{j,1}, \mathsf{cm}_{j,2})$ from P_j, check if $e(pk_j, h) = e(\mathsf{cm}_{j,1}, h) \cdot e(g, \mathsf{cm}_{j,2})$. Only if the check verifies, update $\mathcal{G} \leftarrow \mathcal{G} \cup \{P_j\}$. Store all $\{\mathsf{cm}_j\}_{j \in \mathcal{G}}$.

$\mathsf{Coin}^V[\mathsf{sid}]$**.** Each honest participant P_i performs the following tasks to obtain the coin value for the instance with sid:

- Compute $\sigma_i \leftarrow (H_1(\mathsf{sid})^{\alpha_i}, e(H_1(\mathsf{sid}), sk_i))$ along with a proof $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, H_1(\mathsf{sid}), H_1(\mathsf{sid})^{\alpha_i})$. Multicast (σ_i, π_i).
- Upon receiving (σ_j, π_j) from P_j, check if π_j verifies $\sigma_{j,1}$ and $\mathsf{cm}_{j,1}$. Further, check if $\sigma_{j,2} = e(H_1(\mathsf{sid}), \mathsf{cm}_{j,2}) \cdot e(\sigma_{j,1}, h)$. Upon receiving $t+1$ valid tuples $\{(\sigma_j, \pi_j)\}_{j \in S}$ from distinct nodes in $\mathcal{G}$, compute $\sigma = \prod_{j \in S} \sigma_{j,2}^{\lambda_j}$, where $\lambda_j = \prod_{k \in S \setminus \{j\}} \frac{-k}{j-k}$. Return $r_{\mathsf{sid}} = H_2(\sigma) \in V$.

Fig. 8. The common coin protocol in [16]. It is easy to see the communication complexity for each coin is $O(\lambda n^2)$.

The security of the common coin protocol is based on the co-one-more discrete logarithm (co-OMDL) assumption [18], which we recall in the following definition.

Definition 3 (co-OMDL Assumption). *Let $(\mathbb{G}_1, \mathbb{G}_2, p, g, h)$ be cyclic groups of prime order p. For a positive integer $k \in \mathbb{N}$, the k-co-OMDL assumption states that, for any PPT adversary $\mathcal{A}$,*

$$\Pr\left[\begin{array}{l}(z_1, \ldots, z_k) \leftarrow\!\!\$\ \mathbb{Z}_p^k, \xi_i \leftarrow (g^{z_i}, h^{z_i}) \in \mathbb{G}_1 \times \mathbb{G}_2, \forall i \in [k] \\ (z'_1, \ldots, z'_k) \leftarrow \mathcal{A}^{\mathsf{DL}_g(\cdot)}(\mathbb{G}_1, \mathbb{G}_2, p, g, h, (\xi_i)_{i \in [k]}) : \\ z'_i = z_i, \forall i \in [k]\end{array}\right] \leq \mathsf{negl}(\lambda),$$

where the oracle DL_g on input a group element X in $\mathbb{G}_1$ returns x s.t. $X = g^x$, and $\mathcal{A}$ can access DL_g at most $k-1$ times.

5.2 Warm-up: A Simple Group-Element DKG with Sub-linear Rounds

Figure 9 illustrates our first very simple construction of group-element DKG for Bacho et al.'s common coin protocol [16] with sub-linear rounds from PCSBB and APVSS, which divides the whole network into

a partition $\mathbb{D} := \{\mathcal{D}^{(k)}\}_{k\in[m]}$ with m consortium groups, to take advantage of PCSBB. Every $\mathcal{D}^{(i)}$ in the partition $\mathbb{D}$ contains the (almost) same number η of nodes, and $\mathbb{D}$ does not rely on a trusted setup, as it is an arbitrarily pre-specified parameter that can be hard-coded by the protocol in any deterministic way. In greater detail, the protocol proceeds as follows:

- **Step 1**. For each consortium group $\mathcal{D}^{(k)}$, its member nodes collectively execute an inner-group DKG protocol, which initializes a threshold cryptosystem in a distributed manner across the nodes $\{P_1^{(k)}, \dots P_\eta^{(k)}\}$ in the group. Note that although these inner-group DKG protocols only set up threshold cryptosystems for each consortium group, it suffices to use them to instantiate a PCSBB protocol, thus enabling each honest-majority group to broadcast messages across the whole network efficiently.
- **Step 2**. Then, each node P_i in the group $\mathcal{D}^{(k)}$ samples a random secret s_i and computes a (t, n) PVSS transcript T_i encrypting n secret shares of s_i. Though T_i is a PVSS transcript sharing s_i by encrypting s_i's n secret shares with the public keys of the whole network, P_i only attempts to send T_i to its associated consortium group $\mathcal{D}^{(k)}$. For this purpose, P_i invokes an inner-group BB protocol using T_i as input, and then waits for receiving all $\mathbb{T} := \{T_j\}_{j\in\mathcal{D}^{(k)}}$ from all BBs in its consortium group.
- **Step 3**. Subsequently, every node P_i, as a member of its consortium group $\mathcal{D}^{(k)}$, aggregates $\mathbb{T}$ into an aggregate PVSS transcript $T^{(k)}$, and executes a PCSBB protocol running among the whole network, using $T^{(k)}$ as input. After PCSBB outputs $\mathcal{V}$, P_i aggregates all valid PVSS scripts carried by $\mathcal{V}$ into T. All honest nodes would obtain the same aggregate PVSS transcript T, so they can derive all required public keys and locally decrypt their exclusive secret key shares, thus completing the DKG across the whole network.

As in the recursive construction, contribution records are retained with multiplicity and inputs are aggregated in increasing sender or consortium index order. An input from an individual dealer must pass $\mathsf{PVSS.Vrfy}_1$; an input from consortium $\mathcal{D}^{(j)}$ must pass $\mathsf{PVSS.Vrfy}_{|\mathcal{D}^{(j)}|}$. The BB and PCSBB payload limits use these bounds. The final aggregate has at most $\sum_j |\mathcal{D}^{(j)}| = n < p$ records, so its size remains $O(\lambda n)$ and every retained honest record has nonzero multiplicity modulo p. No CID-ownership check is imposed.

$\mathsf{DKG}[\mathsf{sid}]$. Every $P_i \in \mathcal{D}^{(k)} \subset \mathcal{P}$ performs the following steps:

Step 1: // Each consortium runs an inner-group DKG

activate an inner-group DKG protocol $\mathsf{DKG}[\mathsf{sid}|k]\langle\mathcal{D}^{(k)}\rangle$

Step 2: // Each node broadcasts a PVSS script to its consortium group

$\mathsf{PVSS.Deal}((ek_j)_{j\in[n]}, t, P_i) \to T_i$, which shares a random secret s_i

$\mathsf{BB}[\mathsf{sid}|k]\langle\mathcal{D}^{(k)}, P_i(T_i)\rangle$, which broadcasts T_i across the group $\mathcal{D}^{(k)}$

wait for receiving all $\mathbb{T} := (T_j)_{j\in\mathcal{D}^{(k)}}$ from BBs until next step starts

Step 3: // Every consortium broadcasts its aggregated PVSS to all nodes

$\mathbb{T} \leftarrow (T_j)_{j:\,\mathsf{PVSS.Vrfy}_1(T_j)=1}$; $\mathsf{PVSS.Agg}(\mathbb{T}) \to T^{(k)}$

$\mathsf{PCSBB}[\mathsf{sid}]\langle\{\mathcal{D}^{(1)}, \dots, \mathcal{D}^{(m)}\}, \mathcal{D}^{(k)}(T^{(k)})\rangle \to \mathcal{V}$

$\mathcal{V} \leftarrow (\mathcal{V}[j])_{j:\,\mathsf{PVSS.Vrfy}_{|\mathcal{D}^{(j)}|}(\mathcal{V}[j])=1}$

$\mathsf{PVSS.Agg}(\mathcal{V}) \to T$, $\mathsf{PVSS.PubDriv}(T) \to pk, (pk_j)_{j\in[n]}$, $\mathsf{PVSS.Dec}(dk_i, T) \to sk_i$

output $pk, (pk_j)_{j\in[n]}, sk_i$

//**if** need to flip coins, multicast sk_i and reconstruct sk, **output** $\mathsf{Hash}(sk)$

//or use the DKG output to run the common coin protocol from [16] (Fig. 8)

Fig. 9. A simple sub-linear round DKG protocol for common coin.

Efficiency Analysis. Before analyzing the complexities of our construction, let us first recall the complexities of its building blocks. With $O(\lambda \log n)$-bit MTS signatures and a largest consortium of size η, PCSBB has communication bounded by

$$O\big(nm\ell + \lambda mn^2 \log n + \kappa m(\eta^2 \lambda \log \eta + \eta\lambda \log n)\big)$$

bits and runs for $O(\kappa m)$ rounds. Also, an inner-group DKG protocol costs $O(\eta)$ rounds and $O(\lambda\eta^3)$ communicated bits, and an inner-group BB can attain $O(\eta)$ rounds and $O(\ell\eta + \lambda\eta^2)$ communicated bits, even if instantiating them by classical (adaptively secure) results. Note that the input length ℓ of BB and PCSBB is λn, as it corresponds to the bit length of a PVSS transcript.

Thus, the DKG protocol in Fig. 9 has communication bounded by

$$O\big(\lambda\eta n^2 + \lambda mn^2 \log n + \kappa m(\eta^2 \lambda \log \eta + \eta\lambda \log n)\big)$$

bits and round complexity $O(\eta + m\kappa)$. For $m = \eta = \sqrt{n}$, the communication bound becomes

$$O(\lambda n^{2.5} \log n + \kappa\lambda n^{1.5} \log n),$$

and the round complexity is $O(\kappa\sqrt{n})$.

Security Analysis. The security of our simple sub-linear round DKG protocol described in Fig. 9 can be summarized by the following Lemma.

Lemma 10. *If instantiating our sub-linear DKG protocol in Fig. 9 using the APVSS scheme from [18], it can securely realize the key generation part of the adaptively-secure unique threshold signature scheme from [16], in the presence of a PPT adversary that can adaptively corrupt up to $t < n/2$ nodes.*

Proof (Sketch). Despite the adversary, in the partition $\mathbb{D} := \{\mathcal{D}^{(k)}\}_{k\in[m]}$, there must exist some consortium group $\mathcal{D}^{(h)} \in \mathbb{D}$, such that $\mathcal{D}^{(h)}$ has honest majority, otherwise, the basic honest-majority assumption across the whole network is violated. That said, we can state: (i) all honest nodes in $\mathcal{D}^{(h)}$ must receive from the inner-group BBs the same indexed list $\mathbb{T}$ of PVSS transcripts including those from all honest nodes in $\mathcal{D}^{(h)}$, otherwise, the security of inner-group BB is breached; (ii) PCSBB must deliver the same list $\mathcal{V}$ of results to the whole network, and more importantly, the list $\mathcal{V}$ includes a PVSS transcript that is aggregated from $\mathbb{T}$, otherwise, PCSBB is breached. The honest-majority consortium's output has at most $|\mathcal{D}^{(h)}|$ records, so the length check preserves it. By Lemma 1, aggregating the accepted indexed inputs yields the same valid transcript T and retains the actual records generated by the forever-honest nodes in $\mathcal{D}^{(h)}$, with multiplicity in $\{1, \ldots, n\} \subset \mathbb{Z}_p^*$. Once we prove the above "consensus" part and demonstrate that all honest nodes must obtain an aggregate PVSS transcript (which aggregates at least some honest node's PVSS transcript) despite up to $t < n/2$ adaptive corruptions, the remaining analysis would largely mirror that of the linear-round DKG in [16] and our analysis in § 5.5. We refrain from repeating such analysis and refer interested readers to § 5.5 for details.

5.3 The Recursive Protocol

Deterministic Recursive Partition. In our protocol, the network $\mathcal{P}$ is recursively partitioned into smaller groups according to an arbitrary deterministic rule. Specifically, at the first layer of recursion, $\mathcal{P}$ is divided into two consortiums, $\mathcal{D}^{(1,0)}$ and $\mathcal{D}^{(1,1)}$, such that $\mathcal{P} = \mathcal{D}^{(1,0)} \cup \mathcal{D}^{(1,1)}$, with their sizes approximately equal, i.e., $\lfloor \frac{|\mathcal{P}|}{2} \rfloor$ and $\lceil \frac{|\mathcal{P}|}{2} \rceil$. At the second layer of recursion, each consortium $\mathcal{D}^{(1,k)}$ is further partitioned into $\mathcal{D}^{(2,2k)}$ and $\mathcal{D}^{(2,2k+1)}$. This process continues until the d-th layer, where each consortium $\mathcal{D}^{(d,k)}$ contains a small constant number of nodes. For convenience, we denote the entire network as $\mathcal{D}^{(0,0)} := \mathcal{P}$.

This deterministic recursive partition ensures that if $\mathcal{P}$ has an honest majority, there exists a *forever-honest-majority path*. Specifically, let the adversary corrupt up to $\lfloor \frac{|\mathcal{P}|-1}{2} \rfloor$ nodes, forming the set $\mathcal{C}$. Then, there exists a path $(k_1, \ldots, k_d)$ through the recursive partition tree such that:

$$\mathcal{D}^{(d,k_d)} \subset \mathcal{D}^{(d-1,k_{d-1})} \ldots \subset \mathcal{D}^{(1,k_1)}, \text{ and } |\mathcal{D}^{(r,k_r)} \cap \mathcal{C}| \leq \lfloor \frac{|\mathcal{D}^{(r,k_r)}| - 1}{2} \rfloor, \ \forall r \in [d].$$

This follows directly from the pigeonhole principle.

Building Blocks. We use the following building blocks.

Byzantine Broadcast for base recursion. Since a bottom-level consortium $\mathcal{D}^{(d,k_d)}$ only contains a small number of nodes, it can execute "inefficient" protocols. Specifically, we use an adaptively and concurrently secure deterministic Byzantine broadcast protocol BB (c.f. § 3.5). In particular, we denote the execution of an instance of BB as $\mathsf{BB}[\mathsf{sid}|d|k_d|i]\langle P_i(v), \mathcal{D}^{(d,k_d)}\rangle \to v$, where $(\mathsf{sid}|d|k_d|i)$ is the unique identifier of this instance, and P_i is the sender who broadcasts the message v to the receiver group $\mathcal{D}^{(d,k_d)}$.

PCSBB. We use the PCSBB protocol introduced in Sect. 4. Looking ahead, the protocol PCSBB will be run within every consortium $\mathcal{D}^{(r,k_r)}$ with $r \neq d$, while the two sub-consortiums $\mathcal{D}^{(r+1,k'_{r+1})}$ and $\mathcal{D}^{(r+1,k''_{r+1})}$ serve as the consortiums of the PCSBB instances. We denote the execution of an instance of PCSBB as

$$\mathsf{PCSBB}[\mathsf{sid}|r|k_r]\langle \mathcal{D}^{(r+1,k'_{r+1})}(v'), \mathcal{D}^{(r+1,k''_{r+1})}(v'')\rangle \to \mathcal{V},$$

where $(\mathsf{sid}|r|k_r)$ is the unique identifier of the instance executed within $\mathcal{D}^{(r,k_r)}$, and $\mathcal{D}^{(r+1,k'_{r+1})}$,$\mathcal{D}^{(r+1,k''_{r+1})}$ provide v', v'' as input, respectively; $\mathcal{V}$ is the output.

Running PCSBB within $\mathcal{D}^{(r,k_r)}$ requires that the sub-consortiums $\mathcal{D}^{(r+1,k'_{r+1})}$ and $\mathcal{D}^{(r+1,k''_{r+1})}$ provide *endemic* coins. In our protocol, each sub-consortium will run the one-round common coin protocol from [16] (cf. § 5.1) to provide the endemic coins needed by the PCSBB, while the setup required by the common coin protocol will be provided by our protocol.

APVSS. We use an aggregatable publicly verifiable secret sharing (APVSS) scheme PVSS from [16, 18]. The full definition and construction of this scheme are recalled in § 3.4. In our protocol, we explicitly use the following algorithms of the APVSS scheme.

- $\mathsf{PVSS.Deal}((ek_j)_{j\in\mathcal{D}}, t, P_i) \to T$. It creates a PVSS transcript T under the encryption key set $(ek_j)_{j\in\mathcal{D}}$, while t is the specified threshold, such that using $t+1$ corresponding decryption keys can recover the secret of this transcript.
- $\mathsf{PVSS.Agg}(\{T_j\}_{j\in S}) \to T$. It aggregates a set of PVSS transcripts (under the same set of encryption keys) into one compact transcript T.
- $\mathsf{PVSS.Vrfy}(T) \to 0$ or 1. It validates a transcript.
- $\mathsf{PVSS.Dec}(dk_i, T) \to sk_i$. It decrypts a PVSS transcript T using a decryption key dk_i and yields sk_i, which is a secret share of the secret of this transcript.
- $\mathsf{PVSS.PubDriv}(T)$. It derives the public key and public key shares from T.

As an aggregated PVSS transcript records the exact identities of the source PVSS transcripts, its size grows as the number of source transcripts. To prevent malicious parties from contributing valid yet overly large transcripts, we additionally use $\mathsf{PVSS.Vrfy}_B$, which is $\mathsf{PVSS.Vrfy}$ plus that the number of source transcripts must be bounded by B.

Overview. At a high level, our DKG protocol RDKG proceeds in two steps. First, all nodes in $\mathcal{P}$ agree on an aggregated PVSS transcript, denoted by $T^{(0)}$. Then, each honest node $P_i \in \mathcal{P}$ derives the public keys $pk, (pk_j)_{j\in[n]}$ from $T^{(0)}$ and obtains its secret key sk_i by decrypting $T^{(0)}$. This secret key serves as the output of P_i in the DKG execution.

For security, $T^{(0)}$ must be an aggregation of PVSS transcripts, with at least one transcript generated by a forever-honest node, ensuring that the secret key remains unknown to the adversary. Our protocol adopts a divide-and-conquer strategy to achieve this. Specifically, the top-level consortiums $\mathcal{D}^{(1,0)}$ and $\mathcal{D}^{(1,1)}$ each generate and agree on transcripts $T^{(0)}_{(1,0)}$ and $T^{(0)}_{(1,1)}$, respectively. The final transcript $T^{(0)}$ is obtained by aggregating these two. This strategy is similar to that employed in [16, 46]. Honest nodes in $\mathcal{P}$ must first agree on both $T^{(0)}_{(1,0)}$ and $T^{(0)}_{(1,1)}$ to ensure they can consistently agree on $T^{(0)}$. In [16], this agreement is achieved through two parallel instances of the CSBB protocol [46], which relies on a deterministic Byzantine Agreement protocol, incurring $O(n)$ rounds. Our protocol improves the round complexity by replacing the CSBB instances with our PCSBB protocol, which requires only $O(\kappa)$ rounds.

However, the improved round efficiency of PCSBB relies on efficient generation of endemic coins within each consortium. Therefore, in addition to generating $T^{(0)}_{(1,k)}$ (for $k \in \{0,1\}$), each top-level consortium $\mathcal{D}^{(1,k)}$ must also establish a setup for the common coin protocol (we specifically consider the protocol from [16]) within the consortium. This is achieved by generating and agreeing on another PVSS transcript, $T^{(1)}$, from which honest nodes in $\mathcal{D}^{(1,k)}$ can obtain their secret key shares. Consequently, each top-level consortium $\mathcal{D}^{(1,k)}$ needs to generate and agree on both $T^{(0)}_{(1,k)}$ and $T^{(1)}$. As the recursion proceeds, a consortium $\mathcal{D}^{(r,k_r)}$ at the r-th layer must generate all PVSS transcripts required by its parent consortium, denoted by $T^{(0)}_{(r,k_r)}, \ldots, T^{(r-1)}_{(r,k_r)}$, along with a transcript $T^{(r)}$ to establish the setup for its own endemic common coin protocol.

At each layer $r \in [0,d]$, an honest node P_i must decrypt the PVSS transcript $T^{(r)}$ and use the decryption result to perform common coin. For security, we require that transcripts $T^{(r)}$ and $T^{(r')}$ for $r \neq r'$ be generated with respect to different sets of encryption public keys, i.e., $(ek_i^{(r)})$ and $(ek_i^{(r')})$, such that $ek_i^{(r)} \neq ek_i^{(r')}$ for the same honest node P_i[6]. Therefore, each node needs to generate $d+1 = O(\log n)$ encryption public keys.

Protocol Description. We present the protocol RDKG in Fig. 10. It requires a setup phase in which (1) the public parameters for PVSS, PCSBB, and BB are generated under the transparent setup of § 3 and available to all nodes, and (2) each $P_i \in \mathcal{P}$ runs $(ek_i^{(r)}, dk_i^{(r)}) \leftarrow \mathsf{PVSS.KeyGen}(1^\lambda)$, for $r \in [0,d]$ and generates key pairs for PCSBB and BB, such that all public keys are available to all nodes. Under the setup, the protocol executes as follows:

- **Base case**. For the d-th recursive step, each consortium $\mathcal{D}^{(d,k_d)}$ executes $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$. Specifically, each node P_i computes $d+1$ PVSS transcripts in the base case, each of which encrypts a random secret to be shared among P_i's associated groups $\mathcal{D}^{(0,0)}$, $\mathcal{D}^{(1,k_1)}$, $\cdots$ and $\mathcal{D}^{(d,k_d)}$, respectively. Then, P_i broadcasts these $d+1$ PVSS transcripts across its smallest belonging group $\mathcal{D}^{(d,k_d)}$, then waits for such broadcasted PVSS messages from all nodes in $\mathcal{D}^{(d,k_d)}$. Then, the valid transcripts for the current consortium $\mathcal{D}^{(d,k_d)}$ are aggregated and decrypted, providing a setup for the endemic common coin within $\mathcal{D}^{(d,k_d)}$. The remaining PVSS transcripts are aggregated accordingly, forming a list $\mathbb{T}^{(d,k_d)}$.
- **Recursive execution**. At the r-th recursion, each consortium $\mathcal{D}^{(r,k_r)}$ executes $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, whose execution flow is outlined in Fig. 11. Specifically, the two sub-consortiums, $\mathcal{D}^{(r+1,k_{r+1})}$ and $\mathcal{D}^{(r+1,\overline{k_{r+1}})}$, run $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$, respectively, which establish the setup for endemic common coin within each sub-consortium, as well as produce two lists of PVSS transcripts, $\mathbb{T}^{(r+1,k_{r+1})}$ and $\mathbb{T}^{(r+1,\overline{k_{r+1}})}$. Then, $\mathcal{D}^{(r+1,k_{r+1})}$ and $\mathcal{D}^{(r+1,\overline{k_{r+1}})}$ jointly execute $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ to broadcast the two lists of PVSS transcripts. Once the PCSBB outputs, $P_i \in \mathcal{D}^{(r,k_r)}$ can aggregate its results to obtain a list of further aggregated PVSS transcripts. The aggregate PVSS transcript for $\mathcal{D}^{(r,k_r)}$ can be used to establish a setup for the endemic common coin within $\mathcal{D}^{(r,k_r)}$, which can be used by the $(r-1)$-th recursion to set up its larger PCSBB. And the remaining $(r-1)$ aggregate PVSS transcripts will be passed to the $(r-1)$-th recursive execution, serving as the input of this larger PCSBB.

5.4 Analysis

Efficiency Analysis. Let $n_r = n/2^r$ denote the size of a level-r consortium. In the base case, each party broadcasts $d+1$ PVSS transcripts $\{T_i^{(0)}, \ldots, T_i^{(d)}\}$, whose total size is $O\Big(\lambda \sum_{r=0}^{d} n_r\Big) = O(n\lambda)$. Since broadcasting an ℓ-bit message among η parties costs $O(\eta\ell + \lambda\eta^2 \log \eta)$ bits, a base-case instance satisfies $\mathsf{Cost}_d = O(n_d^2 n\lambda + n_d^3 \lambda \log n_d) = O(n\lambda)$, where $n_d = O(1)$.

At level r, an aggregate contains at most n_r records. Hence its components for all ancestor levels have total size $O\Big(\lambda\Big(\sum_{z=0}^{r} n_z + (r+1)n_r\Big)\Big) = O(n\lambda)$, even when records repeat. A level-r execution consists of

[6] If the same public key is used across layers, the decryption procedure for establishing endemic coins might act as a decryption oracle, which may compromise security.

RDKG[sid]: party P_i with path $\mathcal{D}^{(d,k_d)} \subset \cdots \subset \mathcal{D}^{(1,k_1)} \subset \mathcal{P}$

if sid doesn't contain the fields of r and k, $(r, k_0) \leftarrow (0,0)$; **else** $r \leftarrow \mathsf{sid}.r$ and $k_r \leftarrow \mathsf{sid}.k$

if $r = d$ **then** // **Base case**

$\mathsf{PVSS.Deal}((ek_j^{(0)})_{j\in[n]}, \lfloor \frac{n}{2} \rfloor, P_i) \to T_i^{(0)}, \cdots,$

$\mathsf{PVSS.Deal}((ek_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}, \lfloor \frac{|\mathcal{D}^{(d,k_d)}|}{2} \rfloor, P_i) \to T_i^{(d)}$

$\mathsf{BB}[\mathsf{sid}|d|k_d|i]\langle P_i((T_i^{(d)}, \cdots, T_i^{(0)})), \mathcal{D}^{(d,k_d)}\rangle \to (T_i^{(d)}, \cdots, T_i^{(0)})$

wait for receiving $\{(T_j^{(d)}, \cdots, T_j^{(0)})\}_{j\in\mathcal{D}^{(d,k_d)}}$ from BBs until next step

$S \leftarrow \emptyset$; **for** $j \in \mathcal{D}^{(d,k_d)}$ **if** $\mathsf{PVSS.Vrfy}_1(T_j^{(z)}) = 1, \forall z \in [0,d]$ **then** $S \leftarrow S \cup \{j\}$

$\mathbb{T}^{(d,k_d)} \leftarrow \{\mathsf{PVSS.Agg}((T_j^{(d-1)})_{j\in S}), \cdots, \mathsf{PVSS.Agg}((T_j^{(0)})_{j\in S})\}$

$T^{(d)} \leftarrow \mathsf{PVSS.Agg}((T_j^{(d)})_{j\in S}), sk_i^{(d)} \leftarrow \mathsf{PVSS.Dec}(dk_i^{(d)}, T^{(d)})$

$pk^{(d)}, (pk_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}} \leftarrow \mathsf{PVSS.PubDriv}(T^{(d)})$

output $sk_i^{(d)}, pk^{(d)}, (pk_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}, \mathbb{T}^{(d,k_d)}$

if $r \in \{0, 1, \cdots, d-1\}$ // **Recursive cases**

//RDKG within $\mathcal{D}^{(r+1,k_{r+1})}$, the output except $\mathbb{T}_i^{(r)}$ sets up the endemic coin

$sk_i^{(r+1)}, pk^{(r+1)}, (pk_j^{(r+1)})_{j\in\mathcal{D}^{(r+1,k_{r+1})}}, \mathbb{T}^{(r+1,k_{r+1})}$

$\leftarrow \mathsf{RDKG}[\mathsf{sid}|(r+1)|k_{r+1}]\langle\mathcal{D}^{(r+1,k_{r+1})}\rangle$

//The extra commitment setup for the endemic coin

$\mathsf{Coin.ComSetup}[\mathsf{sid}|(r+1)|k_{r+1}]\langle\mathcal{D}^{(r+1,k_{r+1})}\rangle$

// Broadcast both child transcripts to $\mathcal{D}^{(r,k_r)}$ via PCSBB

$\{\mathbb{T}^{(r+1,k_{r+1})}, \mathbb{T}^{(r+1,\overline{k_{r+1}})}\}$

$\leftarrow \mathsf{PCSBB}[\mathsf{sid}|r|k_r]\langle\{\mathcal{D}^{(r+1,k_{r+1})}(\mathbb{T}^{(r+1,k_{r+1})}), \mathcal{D}^{(r+1,\overline{k_{r+1}})}(\mathbb{T}^{(r+1,\overline{k_{r+1}})})\}\rangle$

//Process the output of PCSBB

parse $\mathbb{T}^{(r+1,k_{r+1})} := \{T_{k_{r+1}}^{(r)}, \cdots, T_{k_{r+1}}^{(0)}\}$, and $\mathbb{T}^{(r+1,\overline{k_{r+1}})} := \{T_{\overline{k_{r+1}}}^{(r)}, \cdots, T_{\overline{k_{r+1}}}^{(0)}\}$

$S \leftarrow \emptyset$; **for** $j \in \{k_{r+1}, \overline{k_{r+1}}\}$

if $\mathsf{PVSS.Vrfy}_{b_j}(T_j^{(z)}) = 1, \forall z \in [0,r]$ **then** $S \leftarrow S \cup \{j\}$

if $r > 0$ **then** $\mathbb{T}^{(r,k_r)} \leftarrow (\mathsf{PVSS.Agg}((T_j^{(z)})_{j\in S}))_{z=r-1,\ldots,0}$

$T^{(r)} \leftarrow \mathsf{PVSS.Agg}((T_j^{(r)})_{j\in S})$

$sk_i^{(r)} \leftarrow \mathsf{PVSS.Dec}(dk_i^{(r)}, T^{(r)})$

$pk^{(r)}, (pk_j^{(r)})_{j\in\mathcal{D}^{(r,k_r)}} \leftarrow \mathsf{PVSS.PubDriv}(T^{(r)})$

output $sk_i^{(r)}, pk^{(r)}, (pk_j^{(r)})_{j\in\mathcal{D}^{(r,k_r)}}$, and if $r > 0$, additionally output $\mathbb{T}^{(r,k_r)}$

Fig. 10. Recursive group-element DKG (RDKG)

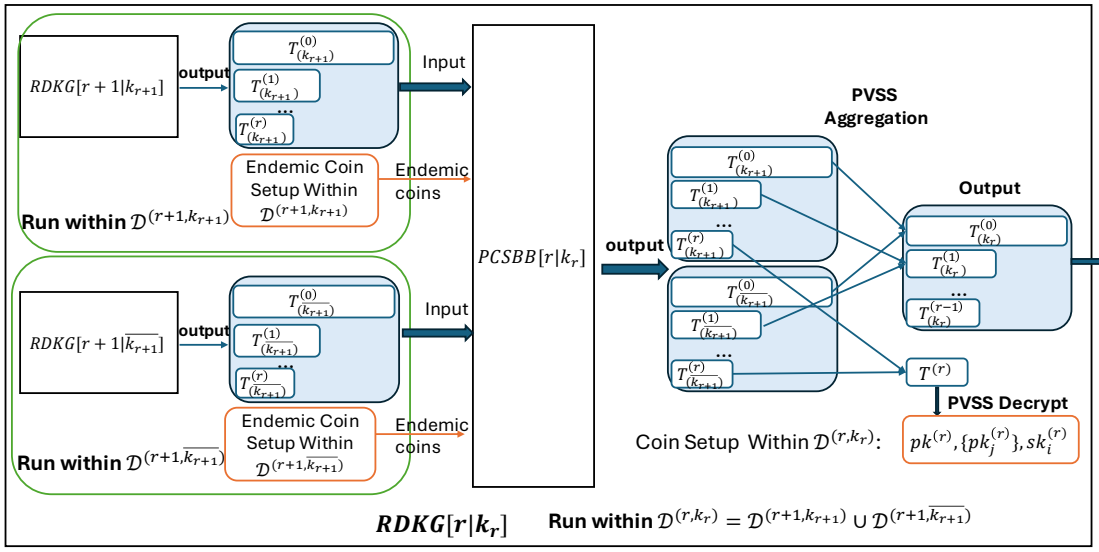

Fig. 11. The execution flow of RDKG at r-th recursion

two child executions and one PCSBB instance with $m = 2$, network size n_r, and input length $O(n\lambda)$. By Theorem 1,

$$\mathsf{Cost}_r \leq 2\mathsf{Cost}_{r+1} + O(\kappa\lambda n_r^2 \log n_r + n_r n\lambda).$$

Therefore, using $n_r = n/2^r$ and $2^d = \Theta(n)$,

$$\mathsf{Cost}_0 \leq 2^d \mathsf{Cost}_d + \sum_{r=0}^{d-1} 2^r O(\kappa\lambda n_r^2 \log n_r + n_r n\lambda) = O(\kappa\lambda n^2 \log n).$$

For round complexity, let N bound the total number of underlying BA and MVBA invocations. Under $N = \mathsf{poly}(\kappa)$, Lemma 2 gives an $O(\kappa)$-round cutoff for each required invocation, except with joint negligible probability. Thus each recursion level takes $O(\kappa)$ rounds, and the recursion depth is $O(\log n)$, giving $O(\kappa \log n)$ rounds overall.

Security Analysis. We establish the following theorem for our RDKG protocol.

Theorem 2. *Under the transparent public-parameter setup of § 3, the SXDH assumption and the co-OMDL assumption, the one-round common coin protocol from [16] (recalled in § 5.1) using* RDKG *as its DKG setup satisfies the availability, agreement, bias-resistance, and unpredictability in the AGM and ROM, against any PPT adversary which can adaptively corrupt up to* $\lfloor \frac{n-1}{2} \rfloor$ *nodes.*

In the following, we sketch our security proof through several key steps, with the full proof given in § 5.5.

Step 1: Generalizing the security proof from [16]. Our objective is to prove the security of the common coin protocol from [16] when using our RDKG protocol as the setup. Ideally, if the common coin protocol has already been proven secure under a DKG with certain security properties, we could establish security by simply demonstrating that RDKG satisfies these required properties. However, [16] only provides a "holistic" security proof, making it necessary for us to delve into the details of the common coin protocol to analyze its security.

Fortunately, we observe that the security proof in [16] can be generalized to accommodate a broader class of DKG protocols with the following structure: (1) The network, through any mechanism, agrees on a valid PVSS transcript w.r.t. the APVSS scheme [18], which is an aggregation of valid PVSS transcripts, containing

at least one generated by a forever-honest node. (2) each node decrypts this aggregated transcript to obtain its secret key share.

We formalize this class of protocols through an *abstract* DKG (described in Fig. 12 in § 5.5). In Lemma 11, we show that, under the co-OMDL assumption (recalled in Definition 3), the common coin protocol is secure under any DKG that conforms to this abstract model, in the AGM and the ROM.

Step 2: RDKG "implements" the abstract DKG. Next, we demonstrate that RDKG serves as an "implementation" of the abstract DKG. Specifically, we show that an adversary $\mathcal{S}$ against the abstract DKG can perfectly simulate an execution of RDKG with any PPT adversary $\mathcal{B}$, such that the simulated execution terminates with an *identical* setup to that of the abstract DKG. Consequently, if $\mathcal{B}$ can successfully attack the common coin protocol under this setup, $\mathcal{S}$ can easily break the security of the common coin under the abstract DKG setup.

The core of this proof is to ensure that in RDKG, the honest nodes in $\mathcal{P}$ agree on a PVSS transcript, which is an aggregation of transcripts containing at least one generated by a forever-honest node. Intuitively, this property is ensured by our RDKG via the following arguments: First, the deterministic recursive partition ensures the existence of a forever-honest-majority path, say $\mathcal{D}^{(d,k_d)} \subset \mathcal{D}^{(d-1,k_{d-1})} \dots \subset \mathcal{P}$. Along this path, each consortium maintains a majority of forever-honest nodes. Under the SXDH assumption, the underlying BB and PCSBB (given proper setup) are secure. Then, ensured by the validity of BB, the PVSS transcripts generated by forever-honest nodes in $\mathcal{D}^{(d,k_d)}$ are received by all honest nodes in $\mathcal{D}^{(d,k_d)}$ and subsequently aggregated into $\mathbb{T}^{(d,k_d)}$. By the validity of PCSBB, $\mathbb{T}^{(d,k_d)}$ is propagated to all honest nodes in $\mathcal{D}^{(d-1,k_{d-1})}$ and aggregated into $\mathbb{T}^{(d-1,k_{d-1})}$. Repeating this argument up the recursion layers, the final transcript $T^{(0)}$ retains the actual records generated by forever-honest nodes. The length bounds never reject an honest child's output, and concatenation preserves every occurrence. Each retained record has multiplicity between one and $n < p$, so repetition cannot cancel the honest contribution modulo p. Lemma 1 establishes the required aggregation invariant.

One subtle issue is that the security of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ depends on the secure execution of the endemic common coin protocol within the honest-majority sub-consortium $\mathcal{D}^{(r+1,k_{r+1})}$. This, in turn, requires that $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ provides a secure DKG setup for common coin within $\mathcal{D}^{(r+1,k_{r+1})}$. We formalize this dependency with a layer-by-layer security analysis.

Defining the security of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$: For the analysis, we define the security of each subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ in Definition 4. In particular, it is secure if, when $\mathcal{D}^{(r,k_r)}$ has an honest majority: (1) All honest nodes within $\mathcal{D}^{(r,k_r)}$ agree on the same tuple $\mathbb{T}^{(r,k_r)}$, which consists of valid PVSS transcripts, and each of them is an aggregate with an occurrence-preserving contribution list containing the actual record of at least one forever-honest node. (2) The setup provided by $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ ensures secure common coin within $\mathcal{D}^{(r,k_r)}$.

Security analysis for the base case $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$: In Lemma 13, we prove that $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ satisfies the security requirements in Definition 4. Specifically, the security of BB ensures that all honest nodes in $\mathcal{D}^{(d,k_d)}$ agree on a set of PVSS transcripts, including those sent by honest nodes. The aggregated transcript contains contributions from honest nodes, and the setup derived from $T^{(d)}$ guarantees secure common coin within $\mathcal{D}^{(d,k_d)}$.

Security analysis for the r-th recursion $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$: In Lemma 14, we prove that if the subroutines $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$ are secure, then $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure. When $\mathcal{D}^{(r,k_r)}$ has an honest majority, at least one sub-consortium, say $\mathcal{D}^{(r+1,k_{r+1})}$, also has an honest majority. This enables secure common coin within $\mathcal{D}^{(r+1,k_{r+1})}$, ensuring the security of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$. Thus, the aggregated PVSS transcripts $\mathbb{T}^{(r+1,k_{r+1})}$ are aggregated into $\mathbb{T}^{(r,k_r)}$, which forms the basis for secure common coin within $\mathcal{D}^{(r,k_r)}$.

By combining Lemma 13 and Lemma 14, we conclude that $\mathsf{RDKG}[\mathsf{sid}]$ establishes a secure DKG setup for common coin within $\mathcal{P}$, completing the proof.

Security justifications for component composition. We now justify the security of multiple subroutines when they are composed inside $\mathsf{RDKG}[\mathsf{sid}]$, following the proof strategy developed in [48]. This follows from the structure of our protocol, where different subroutines are either executed in parallel by disjoint sets of nodes (e.g., $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ and $\mathsf{RDKG}[\mathsf{sid}|r|\overline{k_r}]$) or sequentially composed (e.g., $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$

and $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$). However, the only piece of secret information reused across subroutines is the secret signing key for the underlying MTS scheme (used inside PCSBB). An adversary might thus obtain valid signatures from honest nodes across different subroutines. Nevertheless, since every subroutine has a unique identifier and all messages in a subroutine must begin with this identifier, signatures obtained in one subroutine cannot help the adversary forge signatures in another.

Formally, in Lemma 12, we prove that if a subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure in the stand-alone setting against an enhanced adversary (as defined in Remark 2), it remains secure when composed with other subroutines. This enhanced adversary can control nodes outside the sub-consortium $\mathcal{D}^{(r,k_r)}$ and query a signing oracle for messages not prefixed by $(\mathsf{sid}|r|k_r)$. Since such an adversary can simulate all other subroutines except $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, any attack on the composition can be reduced to the stand-alone case. We apply a similar approach to justify the security of PCSBB and BB when composed with other components.

5.5 Full Security Proof for the Common Coin

This subsection proves Theorem 2. The original security proof in [16] shows the common coin protocol in Fig. 8 is secure under the specific recursive DKG protocol presented in [16]. Since we give an improved recursive DKG (RDKG) protocol in § 5.3 to replace its original DKG, it becomes necessary to analyze its security under our new DKG, despite the generation of same key structure.

Security under Abstract DKG. We proceed with the security proof via the following two steps. First, we generalize the original security proof in [16] to show that the common coin protocol is secure as long as the DKG protocol can be "captured" by the abstract DKG setup we defined in Fig. 12. Next, we show our RDKG in Fig. 10 indeed satisfies this abstraction.

For each honestly generated APVSS transcript, let ρ_i denote the complete internal randomness used by $\mathsf{PVSS.Deal}$. We write $\mathsf{PVSS.Deal}(\cdot;\rho_i)$ for execution with this random tape. Together with the public inputs and the same random-oracle answers, ρ_i determines the dealer's internal state; its contents need not be specified separately. The abstract experiment retains ρ_i and returns it upon corruption, including corruptions after the DKG setup, without assuming erasure.

AbstractGenAPVSS

Initialize $\mathcal{H} \leftarrow [n]$

$\mathsf{Init}^{\mathcal{A}}(n, 1^\lambda) \to (\mathtt{crs}, (ek_i)_{i\in[n]}, (dk_i)_{i\in\mathcal{H}}, \mathsf{state}_{\mathcal{A}})$

for $j \in \mathcal{H}$

  Sample and retain ρ_j

  $\mathsf{PVSS.Deal}((ek_i)_{i\in[n]}, t, P_j; \rho_j) \to (\mathsf{Trans}_j, sk^{(j)})$; let R_j be its sole record

$\mathsf{Trans}^* \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{Corr}}}((\mathsf{Trans}_j)_{j\in\mathcal{H}}), s.t., \mathsf{Check}(\mathsf{Trans}^*) = 1$

return Trans^*

DKGOutput(Trans^*)

return $(\mathsf{PubDriv}(\mathsf{Trans}^*), \{\mathsf{Dec}(ek_i, dk_i, \mathsf{Trans}^*)\}_{i\in\mathcal{H}})$

$\mathcal{O}_{\mathsf{Corr}}(i)$

if $i \in \mathcal{H}$

  $\mathcal{H} \leftarrow \mathcal{H} \setminus \{i\}$

  return $(dk_i, sk^{(i)}, \rho_i)$

Check(Trans^*)

if $\mathsf{PVSS.Vrfy}_n(\mathsf{Trans}^*) = 1$

  $\wedge\ \exists j \in \mathcal{H} : \mathsf{mult}(R_j, \mathsf{Trans}^*) > 0$

  return 1

return 0

Fig. 12. Abstract DKG Setup using the APVSS scheme in § 3.4. It first runs $\mathsf{AbstractGenAPVSS}$ with $\mathcal{A}$ to obtain Trans^* and then outputs $\mathsf{DKGOutput}(\mathsf{Trans}^*)$.

Availability and agreement. We show that when the APVSS transcript Trans^* is valid, the common coin protocol satisfies the availability and agreement. Let $(pk, (pk_i)_{i\in[n]}, (sk_i)_{i\in\mathcal{H}}) \leftarrow \mathsf{DKGOutput}(\mathsf{Trans}^*)$, and

when Trans^* is valid, there exists a t-degree polynomial f, such that $pk_i = g^{f(i)} \in \mathbb{G}_1$ for all $i \in [n]$, $sk_i = h^{f(i)} \in \mathbb{G}_2$ for all $i \in \mathcal{H}$, and $pk = g^{f(0)}$.

First, it is easy to see that during the $\mathsf{ComSetup}$, cm_i produced by an honest participant P_i can pass the verification:

$$\begin{aligned} e(\mathsf{cm}_{i,1}, h)\, e(g, \mathsf{cm}_{i,2}) &= e(g^{\alpha_i}, h)\, e(g, h^{-\alpha_i} h^{f(i)}) \\ &= e(g,h)^{f(i)} = e(pk_i, h). \end{aligned}$$

Therefore, for any honest P_i, its local set $\mathcal{G}$ contains all forever honest participants, which is at least $t+1$.

Then, for every $\mathsf{Coin}^V[\mathsf{sid}]$, the tuple (σ_i, π_i) produced by an honest P_i can pass the verification. Therefore, every honest P_i can receive $t+1$ valid pairs from the nodes in $\mathcal{G}$. Thus, every honest P_i can produce an output in $\mathsf{Coin}^V[\mathsf{sid}]$, which satisfies the availability.

If (σ_j, π_j) passes verification, soundness of Dleq implies, except with negligible probability, that $\sigma_{j,1} = H_1(\mathsf{sid})^{\alpha_j}$ and $\mathsf{cm}_{j,1} = g^{\alpha_j}$ for the same α_j. The commitment check for $P_j \in \mathcal{G}$ then gives $\mathsf{cm}_{j,2} = h^{-\alpha_j} h^{f(j)}$, by non-degeneracy of the pairing. Consequently, the share check implies

$$\begin{aligned} \sigma_{j,2} &= e(H_1(\mathsf{sid}), h^{-\alpha_j} h^{f(j)})\, e(H_1(\mathsf{sid})^{\alpha_j}, h) \\ &= e(H_1(\mathsf{sid}), h)^{f(j)}. \end{aligned}$$

Since the indices in S are distinct and $n < p$, every denominator in $\lambda_j = \prod_{k \in S \setminus \{j\}} (-k)/(j-k)$ is nonzero in $\mathbb{Z}_p$. Lagrange interpolation at zero gives $\sum_{j \in S} \lambda_j f(j) = f(0)$ for every $|S| = t+1$. Thus every such set reconstructs

$$\sigma = \prod_{j \in S} \sigma_{j,2}^{\lambda_j} = e(H_1(\mathsf{sid}), h)^{f(0)},$$

which ensures agreement.

Bias-resistance and unpredictability. We establish the following lemma.

Lemma 11. *Assume that there are a set of polynomially many instances $\{\mathsf{Coin}^V[\mathsf{sid}_k]\}_{k \in S}$ running concurrently under the same abstract DKG setup (c.f. Fig. 12), where $\mathcal{A}$ outputs Trans^* in the $\mathsf{AbstractGenAPVSS}$ subroutine. Let t be the APVSS polynomial degree, so the reconstruction threshold is $q = t+1$. The adversary is algebraic and PPT in the AGM and ROM and corrupts at most $b \le \min\{t, n-t-1\}$ parties over the complete execution. Assume $n < p$ and that the final transcript contains the actual record R_{j^*} of at least one forever-honest party, i.e., $\mathsf{mult}(R_{j^*}, \mathsf{Trans}^*) > 0$. Under the assumptions of Theorem 2, every $k \in S$ satisfies bias-resistance and unpredictability before q distinct parties have either been corrupted or activated in that instance. The bound on b ensures both $b < q$ and at least q forever-honest participants.*

The proof adapts the common-coin reduction of [16]. We track the multiplicity of the designated honest record explicitly; in particular, its coefficient need not be one.

In the following, we show how to slightly modify the original proof for proving the security under the abstract DKG setup.

We use the activation convention from Section 3. At time τ, let $\mathcal{C}(\tau)$ be the current corruption set, $\mathcal{A}_k(\tau)$ the parties activated while honest in instance k, and $a_k(\tau) = |\mathcal{A}_k(\tau) \setminus \mathcal{C}(\tau)|$. Thus

$$a_k(\tau) < q - |\mathcal{C}(\tau)| \quad \Longleftrightarrow \quad |\mathcal{C}(\tau) \cup \mathcal{A}_k(\tau)| < q.$$

This condition is evaluated at the time of the attempted prediction. It counts each party once, even if an activated party is subsequently corrupted, and never becomes true again after the threshold has been reached.

Write $\sigma_k^* = e(H_1(\mathsf{sid}_k), h^{f(0)})$ for the unique reconstructed pairing value and $r_{\mathsf{sid}_k} = H_2(\sigma_k^*) \in V$ for the actual coin output. If H_2 has not been queried at σ_k^*, its value is fresh and uniform. Consequently, a non-negligible distinguishing advantage at an eligible snapshot requires a query at this point, either in the preceding view or during the distinguisher's computation with that frozen view. Guessing among the polynomially many query inputs and target instances gives a polynomial-loss reduction to computing σ_k^*

under the same early condition. This is the auxiliary game below; ordinary guessing of r_{sid_k} has the baseline success probability $1/|V|$ and is not a successful forgery.

Game 0: Run the abstract DKG setup and the common-coin commitment setup. The adversary controls the scheduling of polynomially many coin instances with distinct identifiers $(\mathsf{sid}_k)_{k\in S}$, observes the prescribed messages, and may corrupt parties throughout setup and all subsequent coin instances, subject to the total budget b. Corruption reveals the full retained state, including the dealer randomness ρ_i as specified above. For each instance, record a party in $\mathcal{A}_k$ when its honest share-generation action is activated, before exposing its message to the adversary; record it only once and retain this record even if the message is retracted. Corruptions update $\mathcal{C}$, so $a_k = |\mathcal{A}_k \setminus \mathcal{C}|$ always counts currently honest activated parties.

At any polynomially bounded time τ, the adversary outputs (k, σ) and the experiment stops. It wins exactly when the setup is valid and

$$\begin{gathered} k \in S, \qquad |\mathcal{C}(\tau)| \le b, \qquad a_k(\tau) < q - |\mathcal{C}(\tau)|, \\ \exists j \notin \mathcal{C}(\tau) : \ \mathsf{mult}(R_j, \mathsf{Trans}^*) > 0, \\ \sigma = e(H_1(\mathsf{sid}_k), h^{f(0)}). \end{gathered}$$

The record-inclusion condition is the admissibility requirement of Lemma 11. An invalid target, an exceeded corruption budget, or output at or after the activation threshold loses the game. Denote the winning probability by $\mathsf{Adv}^0_{\mathcal{A}}(\lambda)$. Games 1–6 retain this entire winning predicate, the same activation bookkeeping, and the same corruption budget; they only add the changes explicitly stated below.

Game 1: It is identical to Game 0, except that it adds an additional abort condition. It randomly samples $k^* \leftarrow\!\$\ S$ in the beginning and aborts when $k \neq k^*$ for (k, σ) returned by $\mathcal{A}$. Note that the choice of k^* is independent of $\mathcal{A}$'s view. Therefore, the advantage of $\mathcal{A}$ in this game, $\mathsf{Adv}^1_{\mathcal{A}}(\lambda)$, is at least $\frac{\mathsf{Adv}^0_{\mathcal{A}}(\lambda)}{|S|}$, where $|S|$ is the number of all Coin instances, which is assumed to be polynomial in λ.

Game 2: It is identical to Game 1, except that it reprograms the random oracle H_1 as follows. For input sid_{k^*}, program $H_1(\mathsf{sid}_{k^*}) := \xi_{1,1}$, where $(\xi_1, \ldots, \xi_n)$ is the co-OMDL challenge. For other inputs, randomly sample $\gamma \leftarrow\!\$\ \mathbb{Z}_p^*$, and return g^γ. Note that the standard techniques are applied to ensure the consistency of queries. From the point of view of $\mathcal{A}$, this game is identical to the last game, and thus $\mathsf{Adv}^2_{\mathcal{A}}(\lambda) = \mathsf{Adv}^1_{\mathcal{A}}(\lambda)$.

Game 3: It is identical to Game 2, except that it adds an additional abort condition. It samples $i^* \leftarrow\!\$\ [n]$ at the beginning and aborts if P_{i^*} is corrupted or $\mathsf{mult}(R_{i^*}, \mathsf{Trans}^*) = 0$. The experiment requires at least one forever-honest party whose actual record occurs in the final list; the hidden guess of its index therefore succeeds with probability at least $1/n$. Therefore, the advantage of $\mathcal{A}$ in this game, $\mathsf{Adv}^3_{\mathcal{A}}(\lambda)$, is at least $\frac{\mathsf{Adv}^2_{\mathcal{A}}(\lambda)}{n}$.

Game 4: It is identical to Game 3, except that it adds an additional abort condition. Namely, when the adversary can violate the Sim-Ext security of the SoK in the APVSS scheme, it aborts.

Game 5: It is identical to Game 4, except that it adds an additional abort condition. Namely, when the adversary can violate the soundness of Dleq in the common coin protocol, it aborts.

Game 6: It is identical to Game 5, except that it adds an additional abort condition. Namely, when the adversary can find a collision for H_1, it aborts.

Given the security of the underlying building blocks, it is easy to see the advantage of $\mathcal{A}$ in Game 6, denoted by $\mathsf{Adv}^6_{\mathcal{A}}(\lambda)$, is at least $\frac{\mathsf{Adv}^0_{\mathcal{A}}(\lambda)}{n\cdot|S|} - \mathsf{negl}(\lambda)$. Then, if $\mathsf{Adv}^0_{\mathcal{A}}(\lambda)$ is non-negligible, $\mathsf{Adv}^6_{\mathcal{A}}(\lambda)$ is also non-negligible.

It remains to prove $\mathsf{Adv}^6_{\mathcal{A}}(\lambda)$ is negligible. We adapt the reductions in [16, Pages 24–29], which bound the adversary's advantage in their game $\mathbf{G}_7$ under the co-OMDL assumption (cf. Def. 3), as follows. Our **Game 6** exposes the group descriptions, random-oracle queries, APVSS transcripts, common-coin messages, and the retained state of corrupted parties, including their dealer tapes ρ_i. We explain below how to extend the cited reductions to answer these corruption queries. The other change is how the final Trans^* is generated. In our **Game 6**, the adversary seems to be more powerful as it can specify Trans^*, as long as it satisfies that

$\mathsf{Check}(\mathsf{Trans}^*) = 1$, and that $\mathsf{mult}(R_{i^*}, \mathsf{Trans}^*) > 0$ for a forever-honest P_{i^*}; in their game $\mathbf{G}_7$, the adversary can interfere with the generation of Trans^* by corrupting participants in their DKG protocol. However, after a careful inspection, we can see that the original proof actually works for the seemingly more powerful adversary.

In particular, the original proof gives six PPT algorithms $A_1, \ldots A_6$ for solving the co-OMDL problem, which use four simulation strategies $\{\mathsf{Sim}_1, \ldots, \mathsf{Sim}_4\}$ to invoke a PPT adversary $\mathcal{A}$ against the common coin protocol. It shows that, if an algebraic adversary $\mathcal{A}$ can produce (k, σ) satisfying Game 0's complete winning predicate with non-negligible probability, then at least one of the six algorithms can solve the co-OMDL problem with a non-negligible probability.

We can build PPT algorithms $B_1, \ldots, B_6$ accordingly for solving the co-OMDL problem by using an adversary $\mathcal{A}$ against the common coin protocol under the abstract DKG.

- B_1, B_2 and B_3: Note that A_1, A_2, and A_3 only use the robustness of their DKG protocol, i.e., all honest nodes will output consistent secret key shares at the end of the DKG. In particular, A_1 (resp. A_2 or A_3) applies the simulation strategy Sim_1 (resp. Sim_2 or Sim_3), where the simulation algorithm acts on behalf of all honest nodes during the DKG protocol by honestly following the protocol specification.
  B_1, B_2 and B_3 are identical to A_1, A_2, and A_3, respectively, except that during the DKG part, B_1, B_2 and B_3 honestly generate the PVSS transcripts on behalf of all honest nodes and let $\mathcal{A}$ specify the final Trans^*.

We note that only the simulation strategy Sim_4, used by A_4, A_5, A_6, utilizes "properties" of their DKG protocol.

- Sim_4 [16, Page 26]: On input the co-OMDL challenge $(\xi_1, \ldots, \xi_n)$, it queries DL_g with $(\xi_{t+2}, \ldots, \xi_n)$ and gets $(z_{t+2}, \ldots, z_n)$. Then, during the DKG protocol, it honestly generates the PVSS transcripts on behalf of all honest nodes except P_{i^*} (that contributes to the final aggregated PVSS transcript and remains honest). For P_{i^*}, Sim_4 generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [t]$ (i.e., the $t+1$ coefficients of the polynomial are given by the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). From this, it can generate the commitments and encrypted shares of party P^*'s PVSS transcript by Lagrange interpolation in the exponent and knowledge of the secret keys sk_j of all parties. Since Sim_4 generates the public keys pk_i of honest parties faithfully and the adversary A is algebraic, it outputs the (updated) public keys of corrupt parties as a linear combination of known values, which enables Sim_4 to compute the respective secret keys from this linear combination along with the algebraic representation. In addition, the signature of knowledge w.r.t. $A_0^{(i^*)} = g^{f_{i^*}(0)}$ is generated with the simulation algorithm of the underlying SoK.
- $A_4, A_5,$ and A_6: The three algorithms use Sim_4 to simulate their game $\mathbf{G}_7$ for invoking the adversary $\mathcal{A}$. To convert the forgery (k, σ) outputted by $\mathcal{A}$ into a solution of the co-OMDL challenge, the relevant DKG identity becomes $pk = \xi_{1,1}^{\mu} g^{\alpha}$, where $\mu \in \mathbb{Z}_p^*$ is the known multiplicity of the designated record and the reduction obtains the scalar α, as detailed below.
- $B_4, B_5,$ and B_6: They are almost identical to $A_4, A_5,$ and A_6, respectively, except that during the DKG part they generate all PVSS transcripts of honest nodes like Sim_4 does but let $\mathcal{A}$ to decide the final aggregated transcript Trans^*.
  In **Game 6**, let $R^* = R_{i^*}$ be the designated honest record with constant commitment $\xi_{1,1} = g^{z_1}$. Parse $\mathcal{R}(\mathsf{Trans}^*) = [R_a]_{a=1}^{L}$, with $R_a = (\mathtt{cid}_a, C_a, \varsigma_a)$, and define $I^* = \{a : R_a = R^*\}$ and $\mu = |I^*|$. The exact-record inclusion condition and length bound imply $1 \le \mu \le L \le n < p$. Hence μ^{-1} exists in $\mathbb{Z}_p$, regardless of how the adversary chose the repetitions.
  For every other record, the reduction either already knows its constant exponent because it generated that record honestly, or obtains that exponent using the assumed simulation-extraction property of the SoK. Exact copies of R^* are excluded from extraction and are counted in μ; all other repetitions are counted at every list position. Writing $C_a = g^{\alpha_a}$ for these remaining records gives

$$pk = \prod_{a=1}^{L} C_a = \xi_{1,1}^{\mu} g^{\alpha}, \qquad \alpha = \sum_{a \notin I^*} \alpha_a.$$

Thus, if F is the final sharing polynomial, $F(0) = \mu z_1 + \alpha$, rather than $z_1 + \alpha$. The conversion step must retain this known nonzero coefficient. Explicitly, from a correct target pairing value $\sigma = e(H_1(\mathsf{sid}_k), h^{F(0)})$, the reduction computes

$$\widehat{\sigma} = \left(\sigma\, e(H_1(\mathsf{sid}_k), h)^{-\alpha}\right)^{\mu^{-1}} = e(H_1(\mathsf{sid}_k), h^{z_1}).$$

Likewise, any scalar equation recovering $F(0)$ yields $z_1 = \mu^{-1}(F(0) - \alpha)$. These are post-processing operations in the reduction; the adversary sees the actual aggregate and the actual common-coin execution. They require no additional discrete-logarithm oracle queries. The same conversion applies to the occurrence-weighted linear equations in the cited reduction, with the factor μ explicitly retained. No independence between μ and the adversary's view is assumed.

Dealer randomness on corruption. The added leakage is simulated as follows. Algorithms B_1, B_2, B_3 generate every honest dealer transcript with a sampled tape ρ_i, retain that tape, and return it along with the prescribed keys on corruption. Algorithms B_4, B_5, B_6 do the same for every $i \neq i^*$. Their only transcript generated without a known dealer tape is that of the designated challenge dealer P_{i^*}. By the abort rule introduced in **Game 3**, these reductions abort immediately if P_{i^*} is corrupted, before any corruption response is required. Hence they never need to supply a tape for the simulated proof or the unknown polynomial of that dealer. All non-aborting corruption responses, during or after the DKG setup, disclose the actual tapes used for the corresponding transcripts. These responses have the required joint distribution with the adversary's prior view and require no additional discrete-logarithm queries. Thus the strengthened corruption interface preserves the reductions and the existing loss from guessing a forever-honest contributing dealer.

With this multiplicity normalization in the conversion step of [16], it can be established that one of the six algorithms $B_1, \ldots, B_6$ can solve the co-OMDL problem with a non-negligible probability, if the algebraic adversary $\mathcal{A}$ can produce a valid tuple (k, σ) at a time τ with $a_k(\tau) < q - |\mathcal{C}(\tau)|$, under the same corruption budget. This proves the auxiliary early-computation bound, and the random-oracle argument above gives bias-resistance and unpredictability for $r_{\mathsf{sid}_k} = H_2(\sigma_k^*)$.

Security under Our DKG in § 5. Note that our group-element DKG protocol in Fig. 10 has a recursive construction. At every recursion, every consortium has an intra-consortium DKG setup and runs multiple instances of the one-round common coin protocol within the consortium.

Definition 4 (Subroutine security). *Let $\mathcal{D}^{(r,k_r)}$ be a consortium at the r-th recursion, and let $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ be an instance running within $\mathcal{D}^{(r,k_r)}$. We say that $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ is secure if, when $\mathcal{D}^{(r,k_r)}$ has an honest majority (i.e., at least $\lfloor\frac{|\mathcal{D}^{(r,k_r)}|}{2}\rfloor + 1$ nodes in $\mathcal{D}^{(r,k_r)}$ are forever honest), the following properties hold except with negligible probability.*

- ***Agreement:*** *If honest nodes P_i and P_j in the consortium output $\mathbb{T}_i^{(r,k_r)}$ and $\mathbb{T}_j^{(r,k_r)}$ respectively, then $\mathbb{T}_i^{(r,k_r)} = \mathbb{T}_j^{(r,k_r)}$.*
- ***Validity:*** *Within a predetermined Δ_r rounds, each honest node P_i can output*

$$(sk_i^{(r)}, pk^{(r)}, (pk_j^{(r)})_{j\in\mathcal{D}^{(r,k_r)}}, \mathbb{T}^{(r,k_r)}),$$

where the tuple $\mathbb{T}^{(r,k_r)}$ can be parsed as $\{\mathbf{T}_{k_r}^{(r-1)}, \ldots, \mathbf{T}_{k_r}^{(0)}\}$. For all $z \in [0, r-1]$, it holds that $\mathsf{PVSS.Vrfy}((ek_j)_{j\in\mathcal{D}^{(z,k_z)}}, \mathbf{T}_{k_r}^{(z)}) = 1$. Moreover, there exists a forever-honest node P_{i^*} whose actual native record $R_{i^*}^{(z)}$ occurs in every $\mathbf{T}_{k_r}^{(z)}$. More precisely,

$$1 \leq \mathsf{mult}(R_{i^*}^{(z)}, \mathbf{T}_{k_r}^{(z)}) \leq \mathsf{len}(\mathbf{T}_{k_r}^{(z)}) \leq |\mathcal{D}^{(r,k_r)}| < p.$$

The same inclusion and length bounds hold for the transcript $T^{(r)}$ used to derive the current consortium's keys. The contribution lists may contain repeated CIDs and repeated records.

- ***Secure common coin:*** *If each node $P_i \in \mathcal{D}^{(r,k_r)}$ uses $(pk^{(r)}, (pk_j^{(r)})_{j\in\mathcal{D}^{(r,k_r)}}, sk_i^{(r)})$ as the setup key for the common coin protocol, and there are polynomially many instances of the common coin protocol running concurrently, then every instance satisfies the properties of availability, agreement, bias-resistance, and unpredictability.*

Remark 2 (Enhanced stand-alone security). For ease of analysis, fix a consortium $\mathcal{D} = \mathcal{D}^{(r,k_r)}$ of size N. Its sharing polynomial has degree $\lfloor N/2 \rfloor$, so its coin reconstruction threshold is $q_{\mathcal{D}} = \lfloor N/2 \rfloor + 1$. An enhanced stand-alone adversary observes the public parameters, keys, and prescribed messages of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, and may schedule polynomially many subsequent coin instances. It may adaptively corrupt at most

$$b_{\mathcal{D}} = \left\lfloor \frac{N-1}{2} \right\rfloor$$

parties inside $\mathcal{D}$ over the entire execution, including the coin phase. This is the strict honest-majority bound for both even and odd N. For its current corruption set $\mathcal{C}^{\mathcal{A}}(\tau)$, only $\mathcal{C}_{\mathcal{D}}(\tau) = \mathcal{C}^{\mathcal{A}}(\tau) \cap \mathcal{D}$ counts toward this bound.

The security requirements are:

- Agreement and validity of the DKG hold except with negligible probability. The deadline Δ_r applies to its validity/termination requirement; it does not bound the lifetime of the adversary or subsequent coin instances.
- Coin availability and agreement hold, and the coin value is computationally indistinguishable from an independent uniform element of V at every eligible snapshot, as in Section 3. Here eligibility means

  $$a_{\mathsf{sid}'}(\tau) < q_{\mathcal{D}} - |\mathcal{C}_{\mathcal{D}}(\tau)|,$$

  where $a_{\mathsf{sid}'}(\tau)$ counts activated parties of $\mathcal{D}$ that are still uncorrupted. This requirement covers any polynomially bounded time after setup, including times later than Δ_r. Since $q_{\mathcal{D}} \geq b_{\mathcal{D}} + 1$, it also implies the generic $(N, b_{\mathcal{D}})$ coin guarantee.

The proof uses an auxiliary game: the adversary outputs (sid', σ) at an eligible snapshot with

$$\sigma = e(H_1(\mathsf{sid}'), sk^{(r)}), \qquad r_{\mathsf{sid}'} = H_2(\sigma) \in V,$$

where $sk^{(r)}$ is the group secret obtained by interpolating the secret key shares. The adversary must output the pairing value σ in this auxiliary game. For the actual coin value $r_{\mathsf{sid}'}$, the requirement is negligible distinguishing advantage (and prediction probability at most $1/|V| + \mathsf{negl}(\lambda)$), rather than negligible probability of a correct guess.

The adversary $\mathcal{A}$ is considered *enhanced* in the sense that it can corrupt all nodes in $\mathcal{P} \setminus \mathcal{D}^{(r,k_r)}$, since these nodes are not participants in the execution of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$. Additionally, the decryption keys $\{dk_i^{(r')}\}_{r'\in[0,r-1]}$ of any honest node $P_i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}^{\mathcal{A}}$ are available to $\mathcal{A}$, where $\mathcal{C}_{\mathsf{init}}^{\mathcal{A}}$ denotes the set of nodes initially corrupted by $\mathcal{A}$. Moreover, $\mathcal{A}$ can query a partial signing oracle $\mathcal{O}_{\mathsf{PartSign}}(\cdot)$ associated with the underlying MTS scheme. However, $\mathcal{A}$ is restricted from querying messages that begin with the session identifier "$\mathsf{sid}|r|k_r$".

Remark 3 (Composition Security). To demonstrate the security of the final protocol $\mathsf{RDKG}[\mathsf{sid}]$, we need to establish that the security of each subroutine holds when composed with other subroutines in the protocol execution. Specifically, we consider the following adversary model.

Let $\mathcal{A}$ be a PPT adversary that can adaptively corrupt participants in $\mathcal{P}$ during the execution of $\mathsf{RDKG}[\mathsf{sid}]$ and observe all communication transcripts generated by honest parties. $\mathcal{A}$ can utilize the information learned from any subroutine to influence the behavior of other subroutines.

The security requirements in the composition setting are those of Remark 2: DKG agreement and validity, and the coin properties of Definition 4, with the same local corruption budget and eligible-snapshot condition. The DKG deadline does not restrict subsequent coin-security experiments.

Fortunately, for our protocol, we can show when $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure in the stand-alone setting against an enhanced adversary, it remains secure in the $\mathsf{RDKG}[\mathsf{sid}]$.

Lemma 12. *Assume the network $\mathcal{P}$ runs $\mathsf{RDKG}[\mathsf{sid}]$, and let $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ be a subroutine of it. For every $r \in [0,d]$ and $k_r \in [0, 2^r - 1]$, if $\mathsf{RDKG}[\mathsf{sid}|r|k_r]\langle\mathcal{D}^{(r,k_r)}\rangle$ is secure in the stand-alone setting against any enhanced adversary, then it remains secure in the composition.*

Proof. Let $\mathcal{A}$ be a PPT adversary which violates one of the security properties of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ in the composition. Then, we can build an enhanced PPT adversary $\mathcal{B}$ which violates the security of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ in the stand-alone setting. In particular, $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ takes the public parameter pp as input and provides pp to $\mathcal{A}$. Once $\mathcal{A}$ provides the set of initially corrupted nodes $\mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$, $\mathcal{B}$ corrupts $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}} := \mathcal{C}^{\mathcal{A}}_{\mathsf{init}} \cup (\mathcal{P} \setminus \mathcal{D}^{(r,k_r)})$. Then, $\mathcal{B}$ receives all public keys and the decryption keys $\{dk_i^{(r')}\}_{r' \in [0,r-1]}$ of all $P_i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$, and it generates the key pairs for all $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}} \setminus \mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$. $\mathcal{B}$ provides the public keys of all $P_i \in \mathcal{P} \setminus \mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$ to $\mathcal{A}$. Then, $\mathcal{A}$ returns the public keys for nodes in $\mathcal{C}^{\mathcal{A}}_{\mathsf{init}}$. Finally, $\mathcal{B}$ returns the public keys of all $P_i \in \mathcal{C}^{\mathcal{B}}_{\mathsf{init}}$.

Execution: $\mathcal{B}$ simulates an execution of $\mathsf{RDKG}[\mathsf{sid}]$ (denoted by $\mathsf{Sim}[\mathsf{sid}]$) for $\mathcal{A}$, while engaging in an execution of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ (denoted by $\mathsf{Real}[\mathsf{sid}|r|k_r]$) with the aim of breaking its security. Let $\mathcal{C}^{\mathcal{B}}$ be the set of nodes so-far corrupted by $\mathcal{B}$ in $\mathsf{Real}[\mathsf{sid}|r|k_r]$, and let $\mathcal{C}^{\mathcal{A}}$ be the set of nodes so-far corrupted by $\mathcal{A}$ in $\mathsf{Sim}[\mathsf{sid}]$. $\mathcal{B}$ works as follows during the execution phase.

First, for any $P_j \in \mathcal{C}^{\mathcal{A}} \cap \mathcal{D}^{(r,k_r)}$, $\mathcal{B}$ forwards every message sent by $\mathcal{A}$ on behalf of P_j in $\mathsf{Sim}[\mathsf{sid}]$ to $\mathsf{Real}[\mathsf{sid}|r|k_r]$ (also on behalf of P_j). For any P_j, *s.t.*, $j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ (the honest nodes), $\mathcal{B}$ forwards every message sent by P_j in $\mathsf{Real}[\mathsf{sid}|r|k_r]$ to $\mathsf{Sim}[\mathsf{sid}]$ (also on behalf of P_j).

While the above strategy states how $\mathcal{B}$ simulates messages sent by P_j for $j \in \mathcal{P} \setminus \mathcal{C}^{\mathcal{A}}$ in the subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, however, in $\mathsf{Sim}[\mathsf{sid}]$, $\mathcal{B}$ also needs to simulate messages in subroutines other than $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$. We can classify the messages into two categories: (1) those sent by $P_j \in \mathcal{C}^{\mathcal{B}} \setminus \mathcal{C}^{\mathcal{A}}$ (the nodes corrupted by $\mathcal{B}$ but not by $\mathcal{A}$), and (2) those sent by $P_j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ (the honest nodes in $\mathcal{D}^{(r,k_r)}$).

For (1), $\mathcal{B}$ can simulate these messages by honestly following the protocol specification, as $\mathcal{B}$ knows the secret states of all $P_j \in \mathcal{C}^{\mathcal{B}} \setminus \mathcal{C}^{\mathcal{A}}$. For (2), according to the protocol description, $P_j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ needs to perform actions in $\mathsf{RDKG}[\mathsf{sid}|r-1|k_{r-1}], \ldots, \mathsf{RDKG}[\mathsf{sid}|1|k_1], \mathsf{RDKG}[\mathsf{sid}]$, if $P_j \in \mathcal{D}^{(r-1,k_{r-1})}, \ldots, \mathcal{D}^{(1,k_1)}$. In the following, we illustrate how $\mathcal{B}$ can simulate the actions of P_j in $\mathsf{RDKG}[\mathsf{sid}|r-1|k_{r-1}]$, while the simulation for other subroutines can be done via a similar approach.

- In $\mathsf{RDKG}[\mathsf{sid}|r-1|k_{r-1}]$, following the subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, P_j needs to perform the commitment setup of the common coin instances $\mathsf{Coin.ComSetup}[\mathsf{sid}|r|k_r]$ within $\mathcal{D}^{(r,k_r)}$. Then, in the subroutine $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$, P_j needs to participate in a few common coin instances to generate common coins within $\mathcal{D}^{(r,k_r)}$. $\mathcal{B}$ can simulate these actions in $\mathsf{Sim}[\mathsf{sid}]$ by activating P_j in the corresponding common coin instances in $\mathsf{Real}[\mathsf{sid}|r|k_r]$ and forwarding the messages sent by P_j to $\mathsf{Sim}[\mathsf{sid}]$ (on behalf of P_j).
- In subroutine $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$, the messages sent by $P_i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}^{\mathcal{A}}$ (except the messages for common coin) can be perfectly simulated with the help of a partial signing oracle $\mathcal{O}_{\mathsf{PartSign}}(\cdot)$ w.r.t. the MTS scheme. Therefore, since $\mathcal{B}$ can query the partial signing oracle with messages that do not start with "$\mathsf{sid}|r|k_r$", $\mathcal{B}$ can simulate all messages within $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$.
- Following the subroutine $\mathsf{PCSBB}[\mathsf{sid}|r-1|k_{r-1}]$, P_j needs to process the output of the PCSBB instance, which requires P_j to decrypt the PVSS transcript $T^{(r-1)}$. Since $\mathcal{B}$ is given the decryption keys $\{dk_i^{(r')}\}_{r' \in [0,r-1]}$, $\mathcal{B}$ can perform these actions honestly.

Handle corruption: When $\mathcal{A}$ corrupts P_i in $\mathsf{Sim}[\mathsf{sid}]$, if $i \in \mathcal{D}^{(r,k_r)}$, $\mathcal{B}$ corrupts P_i in $\mathsf{Real}[\mathsf{sid}|r|k_r]$; Otherwise, $\mathcal{B}$ returns the internal states of P_i to $\mathcal{A}$, as this node is controlled by $\mathcal{B}$ in $\mathsf{Sim}[\mathsf{sid}]$.

Output: $\mathcal{B}$ terminates when $\mathcal{A}$ terminates, and it outputs what $\mathcal{A}$ outputs.

It is easy to check that from the point of view of $\mathcal{A}$, the execution of $\mathsf{Sim}[\mathsf{sid}]$ simulated by $\mathcal{B}$ is identical to a real execution of $\mathsf{RDKG}[\mathsf{sid}]$. Therefore, if $\mathcal{A}$ wins with a non-negligible probability, $\mathcal{B}$ can win with a non-negligible probability. □

Lemma 13 (Base Security). *Assume the network $\mathcal{P}$ runs* $\mathsf{RDKG}[\mathsf{sid}]$*, and let* $\{\mathcal{D}^{(d,k_d)}\}_{k_d\in[0,2^d-1]}$ *be all consortiums at d-th recursion (the base case). For each* $k_d \in [0, 2^d - 1]$*, the instance* $\mathsf{RDKG}[\mathsf{sid}|d|k_d]\langle\mathcal{D}^{(d,k_d)}\rangle$ *is secure. I.e., when* $\mathcal{D}^{(d,k_d)}$ *has an honest majority, the instance satisfies agreement and validity, and enables secure common coin, as per Def. 4.*

Proof. We analyze the security of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ against *enhanced* stand-alone adversaries (c.f. Remark. 2), which implies its security when it is composed with other subroutines in $\mathsf{RDKG}[\mathsf{sid}]$, following Lemma 12.

First, we examine the security of $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$ (for $P_i \in \mathcal{D}^{(d,k_d)}$) when composed with other subroutines in $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ under an enhanced adversary. The enhanced adversary can corrupt all parties outside $\mathcal{D}^{(d,k_d)}$, but this does not affect the security of $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$, as those parties are irrelevant to the broadcast instance. Additionally, the adversary can access the signing oracle for the underlying MTS scheme, but only for messages that do not begin with the identifier "$\mathsf{sid}|d|k_d$". Since these signatures cannot be used to forge valid messages for the broadcast instance, the adversary gains no advantage from this capability. Within $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, other subroutines are separate BB instances with different identifiers, which do not interfere with each other. As a result, the security of $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$ is not compromised by its composition with these subroutines. Therefore, $\mathsf{BB}[\mathsf{sid}|d|k_d|i]$ remains secure even when executed within the full protocol and under an enhanced adversary.

Given the security of the underlying BB in the composition, we analyze each property in the following.

Agreement: Ensured by the agreement of BB, all honest nodes output the same tuple $\{(T_j^{(d)}, \cdots T_j^{(0)})\}_{j\in\mathcal{D}^{(d,k_d)}}$. Since $\mathsf{PVSS.Agg}$ is a deterministic algorithm, all participants return the same $\mathbb{T}_i^{(d,k_d)}$

Validity: It is easy to verify that all honest nodes in $\mathcal{D}^{(d,k_d)}$ can terminate in $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ within Δ_{BB} rounds, where Δ_{BB} is the number of rounds of
$\mathsf{BB}[\mathsf{sid}|d|k_d|i]$ (for $P_i \in \mathcal{D}^{(d,k_d)}$). Ensured by the validity of BB, each honest P_i can receive a tuple $(T_j^{(d)}, \cdots T_j^{(0)})$ generated by an honest P_j. Every honest native transcript has one record and passes $\mathsf{PVSS.Vrfy}_1$. Each accepted dealer input has at most one record, so the aggregate has at most $|\mathcal{D}^{(d,k_d)}|$ records. Lemma 1 ensures validity and preserves every honest record, including all of its repetitions. Thus the aggregate has the required form, except with the negligible low-degree-test error.

Secure common coin: Finally, we show the output of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ provides a secure setup for the one-round common coin protocol (cf. Fig. 8) within $\mathcal{D}^{(d,k_d)}$. Namely, let $\mathbf{T}^{(d)}$ be the aggregated PVSS transcript an honest node P_i obtained at the end of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, and let $(pk^{(d)}, (pk_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}, sk_i^{(d)})$ be the public key shares and the secret key share that P_i obtained by decrypting $\mathbf{T}^{(d)}$. Then, the common coin protocol is secure under the set up of $(pk^{(d)}, (pk_j^{(d)})_{j\in\mathcal{D}^{(d,k_d)}}, sk_i^{(d)})$.

We prove this fact by showing that $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ can be "captured" by our abstract DKG in Fig. 12. Specifically, we construct a PPT algorithm $\mathcal{S}$, which is an adversary in the abstract DKG and also *perfectly* simulates an execution of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, such that the transcript $\mathbf{T}^{(d)}$ in the simulated execution is identical to the transcript Trans^* returned by $\mathcal{S}$ in the abstract DKG, which means the simulated execution and the abstract DKG give an identical setup for the common coin protocol. Therefore, if there is a PPT adversary $\mathcal{B}$ which can violate the security of the common coin under the setup of the simulated execution, it can be trivially "translated" into an attack against the common coin protocol under the abstract DKG setup.

For clarity, let $\mathcal{B}$ be a PPT adversary against $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, and we can build $\mathcal{S}$ in Fig. 13. It is easy to verify that, from the point of view of $\mathcal{B}$, the simulated execution is identical to a real execution of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$. Therefore, as we analyzed above, this simulated execution should produce $\mathbf{T}^{(d,k_d)}$, which is a valid aggregated transcript to which a forever-honest P_{i^*} has contributed. Thus, as $\mathcal{S}$ returns Trans^*, it follows that $\mathsf{Check}(\mathsf{Trans}^*) = 1$ as required by the abstract DKG in Fig. 12. Therefore, the abstract DKG and the simulated $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ provide identical setup for the one-round common coin protocol.

Subsequently, if there is a PPT adversary $\mathcal{B}'$ which on the input of $\mathcal{B}$'s internal state can violate the security of the common coin protocol under the setup of the simulated $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$, there is a PPT adversary $\mathcal{S}'$ which on the input of $\mathcal{S}$'s internal state can violate the common coin protocol under the setup of the abstract DKG. By Lemma 11, the common coin protocol is secure under the abstract DKG setup,

conditioned on a forever-honest P_{i^*} having contributed to Trans^*. Therefore, the common coin protocol under the setup of $\mathsf{RDKG}[\mathsf{sid}|d|k_d]$ is secure.

Next, we analyze the security of a recursion $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ for $r < d$, under the condition that its subroutines $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ are secure.

Lemma 14 (Recursion Security). *Assume the network $\mathcal{P}$ runs $\mathsf{RDKG}[\mathsf{sid}]$, and let $\{\mathcal{D}^{(r,k_r)}\}_{k_r\in[0,2^r-1]}$ be all consortiums at r-th recursion, for some $r \in [0, d-1]$. Assuming that its subroutines $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$ are secure in the composition, $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ is secure for $k_r \in [0, 2^r-1]$. I.e., when $\mathcal{D}^{(r,k_r)}$ has an honest majority, the instance satisfies termination, agreement, and validity, and enables secure common coin, as per Definition 4.*

Proof. The subroutine $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ invokes three subroutines, namely $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$, $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$, and $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$. We first examine the security of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ when it is composed of the other two subroutines. Recall the security results of PCSBB in Theorem 1, which states an instance $\mathsf{PCSBB}[\mathsf{sid}']$ satisfies its security properties when (1) one consortium has an honest majority, (2) the common coin instances within the honest-majority consortium is secure, and (3) the adversary cannot forge an MTS signature for messages starting with "sid'". When $\mathcal{D}^{(r,k_r)}$ has an honest majority, one of its two sub-consortiums $\mathcal{D}^{(r+1,k_{r+1})}$ and $\mathcal{D}^{(r+1,\overline{k_{r+1}})}$ must have an honest majority. Given the security of $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$, the common coin instances within the honest-majority sub-consortium are secure. In addition, the information that an adversary can learn from $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ and $\mathsf{RDKG}[\mathsf{sid}|r+1|\overline{k_{r+1}}]$ will not help the adversary to forge an MTS signature for messages starting with "$\mathsf{sid}|r|k_r$".

Given the security of all subroutines when they are composed with each other, we analyze each property in the following. Without loss of generality, we assume that the sub-consortium $\mathcal{D}^{(r+1,k_{r+1})}$ has an honest majority.

Agreement: The agreement follows the agreement property of the underlying PCSBB. In particular, $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ returns the same vector $\mathcal{V}^{(r)}$ to all nodes in $\mathcal{D}^{(r,k_r)}$. Since all honest nodes follow a deterministic rule to parse $\mathcal{V}^{(r)}$ to PVSS transcripts, to verify these transcripts, and to aggregate valid transcripts, all honest nodes have the same aggregated transcripts $\mathbb{T}^{(r,k_r)}$.

Validity: Note that $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$ can terminate within Δ_{r+1} rounds, such that all honest participants have a valid output $\mathbb{T}^{(r+1,k_{r+1})} = \{T^{(r)}_{k_{r+1}}, \cdots, T^{(0)}_{k_{r+1}}\}$. Then, it takes one round to finish the commitment setup for the endemic coin within $\mathcal{D}^{(r+1,k_{r+1})}$. Therefore, the protocol can be configured to start $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ at the beginning of round $\Delta_{r+1}+2$, where the consortium $\mathcal{D}^{(r+1,k_{r+1})}$ (resp. $\mathcal{D}^{(r+1,\overline{k_{r+1}})}$) provides $\mathbb{T}^{(r+1,k_{r+1})}$ (resp. $\mathbb{T}^{(r+1,\overline{k_{r+1}})}$) as the input. Ensured by Theorem 1, $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ can terminate within $\Delta_{\mathsf{PCSBB},r}$ rounds, where $\Delta_{\mathsf{PCSBB},r}$ is a predetermined constant. Hence, honest parties can receive the output of $\mathsf{PCSBB}[\mathsf{sid}|r|k_r]$ by the end of round $\Delta_{r+1}+1+\Delta_{\mathsf{PCSBB},r}$. Ensured by the validity of PCSBB, all honest nodes can receive the input of the honest-majority consortium, i.e., $\mathbb{T}^{(r+1,k_{r+1})}$, and optionally the input from another consortium. Then, the honest nodes can verify the PVSS transcripts and aggregate them accordingly. Ensured by the validity of $\mathsf{RDKG}[\mathsf{sid}|r+1|k_{r+1}]$, all PVSS transcripts in $\mathbb{T}^{(r+1,k_{r+1})}$ are valid and have been contributed by a forever honest node $P_{i^*} \in \mathcal{D}^{(r+1,k_{r+1})}$. The honest child's transcripts satisfy the bound $b_j = |\mathcal{D}^{(r+1,j)}|$ by induction, so they are not removed by the length checks. Every accepted child input has at most b_j records; their concatenation therefore has at most $\sum_j b_j = |\mathcal{D}^{(r,k_r)}|$ records. Lemma 1 ensures that the aggregate is valid and retains the actual records of P_{i^*} with positive multiplicity. Other children may repeat these records, but cannot remove them or cause their multiplicities to vanish modulo p, since the total length is less than p. This proves the induction without a CID-ownership or disjointness check, conditioned on no false acceptance by a low-degree test.

Secure common coin: This can be proved through exactly the same argument in the proof of Lemma 13. In particular, the simulator algorithm $\mathcal{S}$ in Fig. 13 applies to simulating $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ for any $r \in [0, d]$. □

Initial Phase: Once receiving $\texttt{crs}$ from the environment of abstract DKG, $\mathcal{S}$ provides $\texttt{crs}$ to $\mathcal{B}$. Then $\mathcal{B}$ should declare the initially corrupted set $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}}$, such that $\mathcal{C}_{\mathsf{init}} = \mathcal{C}^{\mathcal{B}}_{\mathsf{init}} \cap \mathcal{D}^{(r,k_r)}$ satisfies $|\mathcal{C}_{\mathsf{init}}| \leq \lfloor (|\mathcal{D}^{(r,k_r)}| - 1)/2 \rfloor$. We assume w.l.o.g. that all nodes outside $\mathcal{D}^{(r,k_r)}$ are included in $\mathcal{C}^{\mathcal{B}}_{\mathsf{init}}$, as $\mathcal{B}$ is enhanced. $\mathcal{S}$ then passes $\mathcal{C}_{\mathsf{init}}$ to the environment. Next, $\mathcal{S}$ receives the public encryption keys $(ek_i)_{i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}}$. Then, $\mathcal{S}$ sets $ek_i^{(r)} = ek_i$ for all $i \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$ and generates the key pairs $(ek_i^{(z)}, dk_i^{(z)})$ for all $z \in [0,d], z \neq r$. $\mathcal{S}$ also generates the key pairs w.r.t. the MTS scheme and the digital signature scheme for all nodes in $\mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$. Then, $\mathcal{S}$ provides all public keys of nodes in $\mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$ to $\mathcal{B}$, waits for $\mathcal{B}$ to provide the public keys of $\mathcal{C}_{\mathsf{init}}$, and then provides $(ek_i^{(r)})_{i \in \mathcal{C}_{\mathsf{init}}}$ to the environment.

Input: $\mathcal{S}$ receives $(\mathsf{Trans}_j)_{j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}}$ from the environment of abstract DKG.

Simulating $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$**:** $\mathcal{S}$ simulates the execution as follows.

- During the base case (d-th recursion), $\mathcal{S}$ on behalf of all honest nodes generates the PVSS transcripts. In particular, for $j \in \mathcal{D}^{(r,k_r)} \setminus \mathcal{C}_{\mathsf{init}}$, $\mathcal{S}$ honestly generates the PVSS transcripts $\mathbf{T}_j^{(z)}$ and retains their dealer tapes $\rho_j^{(z)}$, for all $z \in [0,d]$ with $z \neq r$; $\mathcal{S}$ sets $\mathbf{T}_j^{(r)} := \mathsf{Trans}_j$ which it received from the environment of the abstract DKG.
  For all the subsequent operations of $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$, $\mathcal{S}$ acts on behalf of all honest nodes by honestly executing the protocol. We remark that $\mathcal{S}$ can do this, because all secret keys of an honest node P_i, except $(ek_i^{(r)}, dk_i^{(r)})$, are generated by $\mathcal{S}$, and $dk_i^{(r)}$ is not needed to generate any messages in $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$.
- When $\mathcal{B}$ requests to corrupt a previously honest P_i, $\mathcal{S}$ queries $\mathcal{O}_{\mathsf{Corr}}(i)$ and obtains $(dk_i^{(r)}, sk^{(r,i)}, \rho_i^{(r)})$. Using $\rho_i^{(r)}$, it replays the target-layer dealer computation with the same public inputs and random-oracle answers to recover the internal state matching $\mathbf{T}_i^{(r)}$. All other keys, dealer tapes, and simulated local execution records are already known to $\mathcal{S}$. Combining these with $dk_i^{(r)}$, $\mathcal{S}$ returns the full internal state at the time of corruption. The same rule applies to later corruptions after the DKG setup.

Output: When an honest node in the simulated execution obtains the aggregated PVSS transcript $\mathbf{T}^{(r,k_r)}$, $\mathcal{S}$ returns $\mathsf{Trans}^* = \mathbf{T}^{(r,k_r)}$ to the environment of abstract DKG.

Fig. 13. The description of $\mathcal{S}$ which is an adversary in abstract DKG and simulates $\mathsf{RDKG}[\mathsf{sid}|r|k_r]$ for $r \in [0,d]$.

6 Parallel Byzantine Broadcast

We recall the PBC definition below [55].

Definition 5 (PBC). *Let $\mathcal{V}_\ell = \{0,1\}^\ell \cup \{\mathsf{Null}\}$, where Null is a distinguished symbol outside $\{0,1\}^\ell$. A protocol Π involves n parties, each with input $x_i \in \{0,1\}^\ell$, and outputs a vector in $\mathcal{V}_\ell^n$. Faulty senders may have a message or Null in their output entry. We say Π is an (n,t,ℓ) parallel broadcast (PBC), if, in any instance of Π, and for any PPT adversary $\mathcal{A}$ which can corrupt up to t parties, the following properties are ensured except with negligible probability:*

- *Termination: Every honest party P_i eventually outputs a vector $X_i \in \mathcal{V}_\ell^n$.*
- *Agreement: There is one vector $X \in \mathcal{V}_\ell^n$ such that $X_i = X$ for any $i \in \mathcal{H}$, where $\{P_i\}_{i \in \mathcal{H}}$ is the set of honest parties.*
- *Validity: Let $X_i = (x_1^{(i)}, \ldots, x_n^{(i)})$ be the output of an honest P_i. Then, $x_j^{(i)} = x_j$, for any $j \in \mathcal{H}$.*

6.1 Subset Byzantine Broadcast

Following the high-level idea in Section 2.4, the communication cost could be saved if only $O(\kappa)$ randomly selected nodes contribute secrets to the final secret key, which suffices for secrecy as there exists one honest node among the $O(\kappa)$ selected nodes with overwhelming probability. However, if naively using $O(\kappa)$ BB instances for delivering the selected nodes' PVSS transcripts, a rushing adaptive adversary can corrupt the senders and violate robustness. Alternatively, using $O(n)$ BB instances to let all nodes deliver PVSS

transcripts (before selecting $O(\kappa)$-size committee) is robust against rushing adversaries but causes cubic communication.

To address this issue, we introduce *Subset Byzantine Broadcast* (SBC), which begins with a propose phase, in which all nodes disperse their inputs to the network, followed by another delivery phase, where a selected subset of dispersed values will be reconstructed and delivered to the entire network. Intuitively, this simultaneously achieves adaptive security and low communication complexity, because, after the communication-efficient propose phase, it is "too late" for any rushing adversary to prevent the dispersed inputs from being reconstructed. Formally, SBC can be defined as follows.

Definition 6 (Subset Byzantine broadcast). *An (n, t, ℓ)-*SBC *protocol involves n participants $\mathcal{P} = \{P_1, \ldots, P_n\}$ where at most t of them can be corrupted. It syntactically consists of two subsequent phases:*

- SBC.propose*: Every P_i has an ℓ-bit input msg_i to this phase, and would (implicitly) return certain protocol transcripts while P_i terminates.*
- SBC.deliver*: After the propose phase is terminated, each P_i is given the same $\mathbb{S}$ (i.e., a subset of $[n]$) as input for the deliver phase, where $\mathbb{S}$ can be provided by some higher level protocol (e.g., determined by a common coin). At the end of the deliver phase, each P_i will output the corresponding subset of input values $\{\mathsf{msg}'_i\}_{i\in\mathbb{S}}$.*

An (n, t, ℓ)-secure SBC *protocol shall satisfy the following properties.*

- ***Agreement:** Every two honest parties output the same set $\{\mathsf{msg}'_i\}_{i\in\mathbb{S}}$ in the deliver phase.*
- ***Validity:** Let $\mathbb{H}$ be the set of "so-far-honest" parties at the end of the propose phase. Then, with overwhelming probability, for every $j \in \mathbb{H} \cap \mathbb{S}$, we have $\mathsf{msg}'_j = \mathsf{msg}_j$. I.e., the network can deliver the original input value of P_j, if P_j has not been corrupted before the end of the propose phase.*
- ***Termination:** Both propose and deliver phases can terminate.*

The construction. We use the following building blocks: (1) a cryptographic accumulator scheme Acc, (2) an $(n - t, n)$ erasure code scheme EC, and (3) an (n, t, ℓ) Byzantine agreement protocol BA. The first two tools have been introduced in § 3.3. Regarding BA, it allows n nodes to each provide an ℓ-bit input x_i and agree on a common value x. In particular, if all honest nodes input the same x_i, then the agreed output must be $x = x_i$. We recall the formal definition of BA in § 3.5.

Based on these building blocks, we present the SBC construction in Fig. 14, which works as follows. In the propose phase, every node P_i encodes its input msg_i into n fragments $(c_{i,1}, \ldots, c_{i,n})$ using an erasure coding scheme [23], and applies an accumulator scheme (e.g. Merkle tree) to all fragments, so that the fragments correctly committed to the accumulator can later be identified; P_i then sends $c_{i,j}$ and its accumulator witness to each P_j. During the delivery phase, given a subset $\mathbb{S} \subset [n]$, every P_i multicasts the fragments and witnesses received from $\{P_j\}_{j\in\mathbb{S}}$ to the network, so that every node would reconstruct some values associated with $\{P_j\}_{j\in\mathbb{S}}$. Then all participants use $|\mathbb{S}|$ instances of BA [7] to reach a consensus on the reconstructed values. The underlying BA protocol [7] requires common coins, so does our SBC protocol.

Analysis. We establish the following lemma regarding the security and performance of our SBC protocol.

Lemma 15 (SBC). *The protocol in Fig. 14 is an adaptively secure (n, t, ℓ)-*SBC *protocol (cf. Definition 6) for any $t < \frac{n}{2}$, assuming the underlying erasure code and accumulator scheme are secure, and the BA is secure against (strong) adaptive corruption up to t out of n parties. Moreover, for $s = |\mathbb{S}| \geq 1$, the expected communication complexity of* SBC *is*

$$O(n^2|\mathbb{S}|(\lambda + \mathsf{w}) + |\mathbb{S}|n\ell + |\mathbb{S}|\mathsf{BA}_n(\ell)),$$

the expected round complexity under the BA instantiation below is $O(\log(1 + |\mathbb{S}|))$, and the computation cost is $O(|\mathbb{S}|n)$ group operations, where w includes the witness and its $O(\log n)$-bit fragment index, and $\mathsf{BA}_n(\ell)$ denotes an upper bound on the expected communication cost of BA among n participants with ℓ-bit inputs. More specifically, for the BA instantiation in § 3.5, the expected communication complexity becomes

$$O(n^2|\mathbb{S}|(\lambda \log n + \mathsf{w}) + |\mathbb{S}|n\ell).$$

SBC.propose[sid]. Party P_i on input msg_i:

Step 1: // Encode the message and send fragments to different parties

$\mathsf{EC.Enc}(\mathsf{msg}_i) \to (c_{i,1}, \ldots, c_{i,n})$; $\mathsf{Acc.Eval}(ak, \{(j, c_{i,j})\}_{j\in[n]}) \to u_i$

for $j \in [n]$, $\mathsf{Acc.Wit}(ak, u_i, (j, c_{i,j})) \to w_{i,j}$

send $(\mathsf{sid}, \mathsf{propose}, u_i, c_{i,j}, w_{i,j})$ to P_j

Step 2: wait for $(\mathsf{sid}, \mathsf{propose}, u_j, c_{j,i}, w_{j,i})$ from each P_j until Round 1 ends.

then **terminate**

SBC.deliver[sid]. Party P_i on input $\mathbb{S}$:

Step 1: // Echo the fragments from the selected parties

multicast $(\mathsf{sid}, \mathsf{deliver}, \{u_z, (i, c_{z,i}, w_{z,i})\}_{z\in\mathbb{S}})$

Step 2: // Reconstruct the input messages of the selected parties

receive $(\mathsf{sid}, \mathsf{deliver}, \{u_z^{(j)}, (j, c_{z,j}, w_{z,j})\}_{z\in\mathbb{S}})$ from all $P_j \in \mathcal{P}$

for $z \in \mathbb{S}$, **if** $\exists \mathbb{J}_z \subset [n]$ and u_z, s.t. $|\mathbb{J}_z| = n - t \wedge \forall j \in \mathbb{J}_z, u_z^{(j)} = u_z \wedge \mathsf{Acc.Vrfy}(ak, u_z, (j, c_{z,j}), w_{z,j}) = 1$ **then**

$\mathsf{msg}_z^{(i)} = \mathsf{EC.Dec}(\{(j, c_{z,j})\}_{j\in\mathbb{J}_z})$

else $\mathsf{msg}_z^{(i)} = \perp$

Step 3: // reach a consensus on all delivered messages

for $z \in \mathbb{S}$, **invoke** $\mathsf{BA}[\mathsf{sid}||z]$ with the input $\mathsf{msg}_z^{(i)}$, and obtain the output msg_z

output $\{\mathsf{msg}_z\}_{z\in\mathbb{S}}$

Fig. 14. The Subset Byzantine Broadcast protocol using an $(n-t, n)$ erasure code.

When the witness is instantiated by Merkle tree, $\mathsf{w} = \lambda \log n$, hence this further reduces to

$$O(\lambda|\mathbb{S}|n^2 \log n + |\mathbb{S}|n\ell).$$

Proof. Regarding security, *agreement* follows that of BA, and *termination* follows from the synchronous message-delivery bound Δ, and the termination of BA. For *validity*, consider a party $P_i \in \mathbb{H} \cap \mathbb{S}$ with input msg_i. Since party P_i has not been corrupted, even by an adaptive adversary, at the end of the propose phase, it must have correctly generated the n coded fragments from its input msg_i, calculated corresponding witnesses that prove the validity of fragments, and sent them to every $P_j \in \mathcal{P}$. We emphasize that this information is successfully delivered to all honest parties before the deliver phase begins, regardless of whether an adaptive adversary corrupts P_i after the end of the propose phase. At least $n - t$ recipients remain honest throughout the execution. Each retains its authentic fragment and witness even if the dealer is subsequently corrupted, and multicasts them in delivery step 1. Every honest party therefore receives at least $n-t$ authentic fragments with the same accumulator. Accumulator security rules out a different valid fragment at any committed index, except with negligible probability. The $(n-t, n)$ code consequently reconstructs msg_i at every honest party. They all input this value to $\mathsf{BA}[\mathsf{sid}||i]$, whose validity gives the required output.

Next, we analyze the communication complexity of SBC. Each fragment has size $O(\ell/(n-t) + \log n) = O(\ell/n + \log n)$. The field-symbol padding term is absorbed by $\lambda + \mathsf{w}$ under the indexed-witness convention above. In the SBC.propose phase, each party P_i sends out $(\mathsf{sid}, \mathsf{propose}, u_i, c_{i,j}, w_{i,j})$ whose size is $O(\lambda+\mathsf{w}+\ell/n)$ to every $P_j \in \mathcal{P}$. The communication cost of this procedure is $O(n^2(\lambda + \mathsf{w}) + n\ell)$. In the SBC.deliver phase step 1, each party P_i multicasts
$(\mathsf{sid}, \mathsf{deliver}, \{u_z, (i, c_{z,i}, w_{z,i})\}_{z\in\mathbb{S}})$ whose size is $O(|\mathbb{S}| \cdot (\lambda + \mathsf{w} + \ell/n))$. In step 3, each party invokes $|\mathbb{S}|$ BA instances where each BA has an input $\mathsf{msg}_z^{(i)}$ of size $O(\ell)$. Adding the deterministic dispersal costs and the expected BA costs, the expected total communication cost is $O(n^2|\mathbb{S}|(\lambda+\mathsf{w})+n|\mathbb{S}|\ell+|\mathbb{S}|\mathsf{BA}_n(\ell))$. Consider using

the efficient BA instantiation and a Merkle tree instantiation for witness, the expected total communication cost reduces to $O(|\mathbb{S}|n\ell + \lambda|\mathbb{S}|n^2 \log n)$, as stated in Lemma 15. This uses linearity of expectation over the s invocations, whose expected costs are bounded against the common adaptive adversary; it does not assume their executions are independent. A round deadline does not turn this expected communication bound into a bound holding on every execution.

Finally, consider round complexity. For the base BA, let X_j count iterations until successful progress in invocation j. The conditional progress bound in § 3.5 gives $\Pr[X_j > L] \leq 2^{-L}$ in the ideal-component execution. There are at most cs base-BA calls for an absolute constant c, accounting for the extension protocol. Thus

$$\Pr[\max_{j \leq cs} X_j > L] \leq \min\{1, cs\, 2^{-L}\}.$$

Summing this tail over $L \geq 0$ yields $\mathbb{E}[\max_j X_j] = O(\log(1+s))$. The constant number of sequential extension stages and the deterministic propose and reconstruction rounds preserve this bound. Neither identical distributions nor independence are needed.

For at most Q SBC invocations, each selecting at most $s_{\max} \leq n$ indices, let $N_{\mathsf{BA}} = \max\{1, \lceil cQs_{\max} \rceil\}$ bound all base-BA calls. Lemma 2 gives a deadline

$$T_{\mathsf{SBC}} = O(\kappa + \log N_{\mathsf{BA}})$$

from the start of each delivery phase, with joint failure at most $2^{-\kappa} + \nu(\lambda)$. The propose phase costs only constant rounds. Consequently $T_{\mathsf{SBC}} = O(\kappa)$ when $\log(Qn) = O(\kappa)$. For the field-DKG application $s_{\max} \leq 2\kappa$, the sharper condition is $\log(Q\kappa) = O(\kappa)$. Invocations stop sending protocol messages once they terminate; waiting for a predetermined phase deadline requires no extra messages. If the surrounding protocol imposes a cutoff, the expectation bound remains valid because truncation cannot increase communication. The deadline guarantee and the expected communication bound are separate claims. □

6.2 Our PBC Construction

We use the transparent public parameters of § 3. When the common coin is established by our RDKG, the $O(\kappa \log n)$ deadline includes that setup and assumes $\log N = O(\kappa)$ under the global invocation-count convention. With an existing coin setup, the expected-round bound below is unchanged. Given a digital signature PKI setup on top of the exact same setup for a SBC protocol, one can construct an efficient (n, t, ℓ)-PBC protocol, as illustrated by Figure 15. Particularly, the leader election in PBC can be realized with a common coin in a standard way. Each PBC invocation has a unique session identifier sid. For this invocation, define

$$\mu_{\mathsf{pbc}}(j, m) = \mathsf{encode}(\texttt{PBC/input}, \mathsf{sid}, j, m),$$

where encode is an injective, canonical encoding of the tuple, including its field boundaries. The role tag is distinct from every other signing role, including the field-DKG acknowledgement below. Each honest P_j signs only its input for this session, and all parties verify that signature under vk_j.

6.3 Analysis

Lemma 16 (PBC). *Assuming the security of the underlying SBC, the digital signature, and the leader election, the PBC protocol in Fig. 15 is an adaptively secure (n, t, ℓ)-PBC protocol for any $t < \frac{n}{2}$. In addition, under a setup that can provide common coins, its expected communication cost is $O(\kappa n^2 \ell + \lambda \kappa n^2 \log n)$, and its expected round complexity is $O(\log \kappa)$.*

Proof. Regarding security, *agreement* follows the agreement property of SBC, and all honest parties should see the same verification outcomes in step 5. *Termination* follows from the synchronous message-delivery bound Δ, and the termination of SBC, and the Election procedure. Let $\mathbb{H}_r$ be the set of "so-far-honest" parties at the end of step r. Note that $\mathbb{H}_5$ denotes the so-called "honest parties".

PBC[sid]. Each P_i, on input msg_i, performs the following steps:

Step 1:// multicast the signed input

$\mathsf{DS.Sign}(vk_i, sk_i, \mu_{\mathsf{pbc}}(i, \mathsf{msg}_i)) \to \sigma_i$; **multicast** $(\mathsf{sid}, \text{INPUT}, (\mathsf{msg}_i, \sigma_i))$

Step 2:// propose all collected inputs to SBC

receive $(\mathsf{sid}, \text{INPUT}, (\mathsf{msg}_j, \sigma_j))$ from all other P_j

For missing or malformed inputs, set $(\mathsf{msg}_j, \sigma_j) = (\mathsf{Null}, \perp)$

Concatenate: $\mathsf{MSG}_i = ((\mathsf{msg}_j, \sigma_j))_{j\in[n]}$

invoke $\mathsf{SBC.propose}[\mathsf{sid}]$ with input MSG_i, and wait until it terminates

Step 3:// decide which κ messages to deliver

$\mathsf{Election} \to \mathbb{S} \subset [n]$, s.t. $|\mathbb{S}| = \kappa$

Step 4: // deliver elected messages

invoke $\mathsf{SBC.deliver}[\mathsf{sid}]$ with the input $\mathbb{S}$, and obtain the output $\{\mathsf{MSG}_z\}_{z\in\mathbb{S}}$

Step 5:// output according to the delivered messages

for $j \in [n]$, initialize $\mathsf{output}_j = \mathsf{Null}$.

for $z \in \mathbb{S}$ in increasing order, parse $\mathsf{MSG}_z = ((\mathsf{msg}'_j, \sigma'_j))_{j\in[n]}$

Skip malformed vectors; skip entries whose message is not in $\{0,1\}^\ell$

for $j \in [n]$, **if** $\mathsf{DS.Vrfy}(vk_j, \mu_{\mathsf{pbc}}(j, \mathsf{msg}'_j), \sigma'_j) = 1$ and $\mathsf{output}_j = \mathsf{Null}$ **then**

$\mathsf{output}_j = \mathsf{msg}'_j$

output $(\mathsf{output}_j)_{j\in[n]}$

Fig. 15. The Parallel Byzantine Broadcast protocol.

By standard analysis, we know that $\mathbb{H}_3 \cap \mathbb{S} \neq \emptyset$ with overwhelming probability $1 - 2^{-\kappa}$ with $|\mathbb{S}| = \kappa$. Without loss of generality, we assume there exists $P_j \in \mathcal{P}$ such that $P_j \in \mathbb{H}_3 \cap \mathbb{S}$. Observe that all honest parties in $\mathbb{H}_5$ will multicast its input msg with a valid digital signature in step 1. Since $P_j \in \mathbb{H}_3$ has not been corrupted by the end of step 3, it must have received all INPUT messages from $\mathbb{H}_5$ and formed the concatenation MSG_j. Specifically, for all $P_k \in \mathbb{H}_5$, $\mathsf{MSG}_j[k] = (\mathsf{msg}_k, \sigma_k)$. By the validity of SBC, at the end of step 4, all parties in $\mathbb{H}_5$ must have obtained MSG_j in the output of $\mathsf{SBC.deliver}$. Therefore, in step 5, for any $P_k \in \mathbb{H}_5$, P_k must be able to correctly parse $\mathsf{MSG}_j = ((\mathsf{msg}'_q, \sigma'_q))_{q\in[n]}$, and for all $P_q \in \mathbb{H}_5$, the signature validation $\mathsf{DS.Vrfy}(vk_q, \mu_{\mathsf{pbc}}(q, \mathsf{msg}'_q), \sigma'_q) = 1$ must pass.

It remains to show that an earlier accepted value for an honest sender $P_q \in \mathbb{H}_5$ cannot differ from msg_q. Step 5 accepts such a value m only with a signature under vk_q on $\mu_{\mathsf{pbc}}(q, m)$. Since P_q remains honest throughout this invocation, it signs only $\mu_{\mathsf{pbc}}(q, \mathsf{msg}_q)$ in this role and session. If $m \neq \mathsf{msg}_q$, the accepted signature is therefore a forgery, except with negligible probability. The injective encoding and distinct role and session fields ensure that signatures from other invocations or protocols do not sign this same string. Consequently, every honest P_k outputs msg_q for every honest P_q, which establishes validity.

Finally, we investigate the round complexity. By Lemma 15, step 4 incurs $O(\log |\mathbb{S}|) = O(\log \kappa)$ round in expectation. Since other steps incur only constant round complexity, the total expected round complexity is $O(\log \kappa)$. □

7 Field-element DKG

Though the DKG scheme in § 5 achieves nearly optimal communication and round complexities, it however only generates secret keys as Elliptic Curve group elements, which is incompatible with the majority of DLog-based cryptosystems where the secret key is in a scalar field $\mathbb{F}$. To close this gap, in this section, we

present a DKG protocol for field-element secrets, where our group-element DKG only serves as a setup for a common coin protocol. Moreover, for $2\kappa \leq n$, we prove robustness and oracle-aided algebraic simulatability [17] against strongly adaptive adversaries with memory erasures, and full secrecy [56] against static adversaries, with the standard field-element key structure; these security guarantees are stated precisely in Theorem 3.

7.1 Definition

We recall the definition of a DKG for the standard key structure of discrete-logarithm cryptosystems, where the key pair (pk, sk) is generated as follows: first uniformly sample sk from the scalar field $\mathbb{F}$, and then compute $pk = g^{sk} \in \mathbb{G}$, where $g \in \mathbb{G}$ is a generator of the group $\mathbb{G}$.

Syntax. An (n, t)-DKG protocol, denoted as Π_{DKG}, involves n parties $\mathcal{P} = (P_1, \ldots, P_n)$. After its execution, each P_i outputs a public key $pk \in \mathbb{G}$, a list of public key shares $(pk_i)_{i\in[n]} \in \mathbb{G}^n$, and a secret key share $sk_i \in \mathbb{F}$. The protocol is also accompanied by an initial phase and a reconstruction algorithm:

- $\mathsf{Init}(1^\lambda, n)$. It generates the common reference string (CRS)[7] $\mathtt{crs}$ and establishes the PKI for all participants in $\mathcal{P}$.
- $\mathsf{Rec}((i, sk_i)_{i\in\mathbb{I}})$. Given a set of $t+1$ secret key shares as input, the algorithm outputs the secret key $sk \in \mathbb{F}$ corresponding to $pk \in \mathbb{G}$, such that $pk = g^{sk}$.

Regarding security, we consider robustness, oracle-aided algebraic simulatability, and full secrecy, which are defined as follows.

Robustness. Against any PPT adversary adaptively corrupting at most t parties, the following properties hold jointly except with probability $\nu(\lambda) + 2^{-\Omega(\kappa)}$, where ν is negligible. All honest parties terminate with the same public key pk and public shares $(pk_i)_{i\in[n]}$. Each honest P_i holds a share satisfying $pk_i = g^{sk_i}$. There exists a single $sk \in \mathbb{F}$ such that $pk = g^{sk}$ and, simultaneously for every set $\mathbb{I}$ of $t+1$ honest party indices,

$$\mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}}) = sk.$$

Full Secrecy. It captures that during the execution of Π_{DKG}, the adversary learns nothing about the secret key beyond the final public key and cannot bias the key distribution. In particular, Π_{DKG} satisfies *full secrecy*, if for any PPT adversary $\mathcal{A}$ which corrupts up to t nodes, there is a PPT simulator $\mathsf{Sim}^{\mathcal{A}}$, which, on input a uniformly sampled public key $pk \leftarrow\!\!\$\ \mathbb{G}$, outputs a simulated view $\mathsf{sview}_{\mathcal{A}}$, which is computationally indistinguishable from the adversary $\mathcal{A}$'s view $\mathsf{view}_{\mathcal{A}}$ in a real execution of Π_{DKG} that ends with pk as its public key output. Here $\mathsf{view}_{\mathcal{A}}$ includes all public messages and the internal states visible to $\mathcal{A}$.

DKG Oracle-aided algebraic simulatability To capture the adaptive security of many classical DKG protocols, Bacho and Loss [17] introduced *oracle-aided algebraic simulatability*, where the simulator is expected to simulate the execution view with the help of limited algebraic accesses to a DLog oracle. As shown in [17], a DKG with this property is at least sufficient for the adaptive threshold BLS.

Π_{DKG} realizes k-oracle-aided algebraic simulatability if for every PPT adversary $\mathcal{A}$ that adaptively corrupts at most t parties, there exists a PPT simulator $\mathsf{Sim}^{\mathcal{A}}$ which on input $\zeta = (g^{z_1}, \ldots, g^{z_k}) \in \mathbb{G}^k$, can query the DLog oracle $\mathsf{DL}_g(\cdot)$ for at most $k-1$ times, and simulate an execution of Π_{DKG} for $\mathcal{A}$. In particular,

- **Algebraic oracle queries:** When $\mathsf{Sim}^{\mathcal{A}}$ queries $\mathsf{DL}_g(\cdot)$ with a group element g', it must provide the algebraic expression of g' in terms of $(g, g^{z_1}, \ldots, g^{z_k})$, *i.e.*, a vector $(\hat{a}, a_1, \ldots, a_k) \in \mathbb{F}^{k+1}$ such that

$$g' = g^{\hat{a}} \prod_{j\in[k]} (g^{z_j})^{a_j}.$$

The oracle will return $a \in \mathbb{F}$ s.t. $g' = g^a$.

[7] Our construction uses the transparent public parameters and URS of § 3, with no secret setup trapdoor; the generic syntax permits a CRS.

- **Indistinguishable simulation:** Denote by $\mathsf{view}_{\mathcal{A},y,\Pi}$ the view of $\mathcal{A}$ in an execution of Π conditioned on all honest parties outputting $pk = y$. Denote by $\mathsf{view}_{\mathcal{A},\zeta,y,\mathsf{Sim}}$ the view of $\mathcal{A}$ when interacting with Sim on input ζ, conditioned on Sim outputting $pk = y$. Then, for all y and all ζ, $\mathsf{view}_{\mathcal{A},y,\Pi}$ and $\mathsf{view}_{\mathcal{A},\zeta,y,\mathsf{Sim}}$ are computationally indistinguishable.
- **Invertible simulatability matrix:** Let g_i denote the i-th query by Sim to $\mathsf{DL}_g(\cdot)$. Let $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ be the corresponding algebraic coefficients of g_i, *i.e.*, $g_i = g^{\hat{a}_i} \prod_{j=1}^{k} (g^{z_j})^{a_{i,j}}$ and set $(\hat{a}_0, a_{0,1}, \ldots, a_{0,k})$ as the algebraic coefficients of pk. Then, the following matrix over $\mathbb{F}$ is invertible

$$L := \begin{pmatrix} a_{0,1} & a_{0,2} & \cdots & a_{0,k} \\ a_{1,1} & a_{1,2} & \cdots & a_{1,k} \\ \vdots & \vdots & & \vdots \\ a_{k-1,1} & a_{k-1,2} & \cdots & a_{k-1,k} \end{pmatrix}.$$

7.2 The Construction

Building Blocks. Our field-element DKG uses the following components.

Scrape's low degree test. Following [39, 70], we use a variant of Pedersen's polynomial commitment. Particularly, to commit a t-degree polynomial $f \in \mathbb{F}[X]$, we randomly sample another t-degree polynomial $f' \in \mathbb{F}[X]$, and compute the commitment as $(g^{f(1)}h^{f'(1)}, \ldots, g^{f(n)}h^{f'(n)})$, where $n > t+1$ is an integer, and g and h are two uniformly sampled distinct generators of $\mathbb{G}$. Such a polynomial commitment admits the efficient low-degree test due to Scrape [29]. For ease of use, we formulate the next two algorithms for verifying if a vector of n group elements $(C_1, \ldots, C_n)$ commits a t-degree polynomial.

- $\mathsf{PC.VkGen}(n, t, \mathbb{F})$. Uniformly sample $q(X) \in \mathbb{F}[X]$ whose degree is at most $(n-t-2)$, and return $\mathsf{pvk} = (\mathsf{pvk}_j)_{j\in[n]}$ where $\mathsf{pvk}_j = \frac{q(j)}{\prod_{i=1, i\neq j}^{n}(j-i)}$.
- $\mathsf{PC.Vrfy}(\mathsf{pvk}, (C_1, \ldots, C_n))$. Return 1, if $\prod_{j=1}^{n}(C_j)^{\mathsf{pvk}_j} = \mathbf{1}_{\mathbb{G}}$; Otherwise, 0.

The effectiveness of the test is summarized by the following lemma from [39, 70].

Lemma 17 (borrowed from [39, 70]). *Let g and h be two distinct generators for a group $\mathbb{G}$ with the scalar field $\mathbb{F}$. Let $pvk \leftarrow \mathsf{PC.VkGen}(n, t, \mathbb{F})$. Then, for any $(C_1, \ldots, C_n) \in \mathbb{G}^n$, if $\mathsf{PC.Vrfy}(pvk, C_1, \ldots, C_n) = 1$, then*

$$\Pr[\exists t\text{-degree } f, f' \in \mathbb{F}[X],\ s.t.\ C_i = g^{f(i)}h^{f'(i)}, \forall i \in [n]] \geq 1 - \frac{1}{|\mathbb{F}|}.$$

NIZK. We use the ROM argument of knowledge from Definition 1 for

$$R\big(x; (sk, r)\big) = 1 \quad \Longleftrightarrow \quad \mathsf{cm} = g^{sk}h^{r} \ \wedge\ pk = g^{sk}, \qquad x = (\mathsf{sid}, i, \mathsf{cm}, pk, g, h). \tag{6}$$

Here $\mathbb{F} = \mathbb{Z}_p$, $\mathbb{G}$ has prime order $p = 2^{\Omega(\lambda)}$, and g, h are the fixed nonidentity generators. The session sid and share index i are public context. All inputs have canonical, injective encodings; group membership, the prescribed generators, and canonical field encodings are checked. Let H_{zk} be a domain-separated random oracle to $\mathbb{F}$, independent of the coin and signature oracles. We abbreviate calls for the statement above by $\mathsf{NIZK.Prove}[\mathsf{sid}, i]((\mathsf{cm}, pk), (sk, r))$ and $\mathsf{NIZK.Vrfy}[\mathsf{sid}, i]((\mathsf{cm}, pk), \pi)$.

- $\mathsf{NIZK.Prove}(x; (sk, r))$: sample $a, b \leftarrow\!\!\$\ \mathbb{F}$, set $A = g^a$, $B = h^b$, and $c = H_{\mathsf{zk}}(x, A, B)$. Return $\pi = (z_1, z_2, c)$, where $z_1 = a + c\,sk$ and $z_2 = b + c\,r$.
- $\mathsf{NIZK.Vrfy}(x, \pi)$: after the above syntax checks, set $U = \mathsf{cm}/pk$ and accept exactly when

$$c = H_{\mathsf{zk}}\big(x, g^{z_1}pk^{-c}, h^{z_2}U^{-c}\big).$$

- $\mathsf{NIZK.SimProve}(x)$: sample $c, z_1, z_2 \leftarrow\!\!\$\ \mathbb{F}$ and set $A = g^{z_1} pk^{-c}$, $B = h^{z_2} U^{-c}$. If (x, A, B) already has an oracle-table entry, abort this simulation; otherwise set $H_{\mathsf{zk}}(x, A, B) = c$, record $(x, (z_1, z_2, c))$ in $\mathcal{T}$, and return (z_1, z_2, c). Other fresh oracle queries receive independent uniform answers, and repeated queries receive the recorded answer.

Lemma 18 (Security of the share proof). *The above proof has perfect completeness, adaptive multi-theorem zero knowledge, and weak simulation extractability in the classical ROM, with respect to the specified simulator.*

Proof. Completeness follows from $g^{z_1} pk^{-c} = g^a$ and $h^{z_2} U^{-c} = h^b$. For any prescribed challenge c, sampling z_1, z_2 uniformly and reconstructing (A, B) produces exactly the honest interactive transcript distribution. This proves special honest-verifier zero knowledge. For each simulation query, (A, B) is uniform in $\mathbb{G}^2$, even conditioned on the preceding oracle table. With at most q_H external oracle queries and q_S simulated proofs, the probability of a programming collision is at most $q_S(q_H + q_S)/p^2$. Outside this event, the simulated and honest proof distributions coincide. A union bound therefore gives multi-theorem zero knowledge for adaptive requests.

For fixed (x, A, B, c) there is at most one accepting response $(z_1, z_2) \in \mathbb{F}^2$: exponentiation by either nonidentity generator is injective. Moreover, from two accepting interactive transcripts with the same (x, A, B) and distinct c, c', compute

$$sk = \frac{z_1 - z_1'}{c - c'}, \qquad r = \frac{z_2 - z_2'}{c - c'}. \tag{7}$$

Dividing their verification equations gives $pk = g^{sk}$ and $U = h^r$. Thus the recovered pair is a witness for x. This establishes 2-special soundness and unique responses. The first message is uniform over $\mathbb{G}^2$ and has min-entropy $2 \log p$, meeting the theorem's non-triviality condition. Together with the exponentially large challenge space and special honest-verifier zero knowledge, these properties meet the hypotheses of [43, Theorem 3], which gives Definition 1 for the canonical simulator above.

Concretely, a fresh accepted proof cannot use a simulator-programmed point: equality of that point fixes x, A, B, c, and unique responses would make it an exact copy in $\mathcal{T}$. Except for a challenge guess of probability at most $1/p$, its verification point is therefore an ordinary adversarial oracle query. The extractor replays the adversary with the same coins and the same oracle history before that query, changes its challenge answer, and consistently simulates the remaining oracle calls. A successful fork fixes x, A, B and supplies $c \neq c'$, so Eq. (7) recovers the opening. The forking bound in the cited theorem is inverse-polynomial whenever fresh acceptance has non-negligible probability; guessing and programming-collision errors are negligible. The inclusion of the entire statement and context in the hash is essential to fixing the extracted statement.

Use in the DKG simulations. Both simulators invoke this same $\mathsf{SimProve}$ for each so-far-honest party's Step 7 proof, using the actual $(\mathsf{sid}, i, \mathsf{cm}_i, pk_i, g, h)$ and a single shared H_{zk} table. The adversary may query this oracle before and after any simulated proof. If a party is later corrupted, its witness is obtained from the retained shares as prescribed by the DKG simulator; the prover coins consistent with its proof are then $a = z_1 - c\, sk_i$ and $b = z_2 - c\, r_i$. Thus the proof simulation also supplies consistent retained prover state, without requiring an additional erasure. Reusing an exact simulated proof under the same context preserves its already prescribed public key share; using it under another context is a fresh statement–proof pair.

Extraction is used only in the security reduction after a candidate inconsistent share is identified. It is not a step of either DKG simulator. In particular, the oracle-aided simulator's DLog-query budget is not enlarged by extraction: the separate reduction to Pedersen binding can sample the challenge coefficients with known exponents, reproduce the same distribution of the simulated execution, and answer the symbolic DLog queries itself in each replay. This argument requires that the reduction simulate the whole environment, including corruption replies; treating the external DLog oracle as rewindable would not be justified.

Consensus and Leader Election Components. We use SBC (described in Fig. 14) as the consensus component. To provide common coins used in SBC, our group-element DKG protocol RDKG (cf. Fig. 10 in Section 5) is used to establish the setup of one-round common coin protocol from [16] (described in Fig. 8).

The leader election subroutine $\mathsf{Election}^{[n]^{2\kappa}}$ samples a uniformly random subset of $m = 2\kappa$ distinct indices in $[n]$; we assume $2\kappa \le n$ for this fixed-size election. It is realized from the same common coin protocol with output domain $\binom{[n]}{m}$, the set of all m-subsets of $[n]$. We write RO for this election instance's domain-separated final random oracle, so that $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma)$ is the elected subset. In addition, we assume a standard digital signature scheme $\mathsf{DS} = \{\mathsf{DS.KeyGen}, \mathsf{DS.Sign}, \mathsf{DS.Vrfy}\}$ with existential unforgeability under chosen message attacks.

Each field-DKG invocation has a unique session identifier sid. Within this invocation, let

$$\mu_{\mathsf{ack}}(d, j, C) = \mathsf{encode}(\texttt{FDKG/share-ack}, \mathsf{sid}, d, j, C),$$

where encode is an injective, canonical tuple encoding with explicit field boundaries and a canonical encoding of C. The tag identifies this signing role and is distinct from all other protocol roles. The signature by recipient P_j thus acknowledges the opening of commitment C from dealer P_d in this session. For use in Fig. 16, define $\mathsf{CertVrfy}(d, j, C, c) = 1$ precisely when either c is an opening (a, b) with $C = g^a h^b$, or c is a signature σ with

$$\mathsf{DS.Vrfy}(\mathsf{DS}.vk_j, \mu_{\mathsf{ack}}(d, j, C), \sigma) = 1.$$

The two certificate formats are distinguished by explicit type tags, suppressed in the figures; malformed certificates are rejected. The session argument is implicit in both μ_{ack} and $\mathsf{CertVrfy}$. Honest recipients acknowledge at most one valid opening per dealer in a session and retain it, indexed by (sid, d, j, C), until Step 7. For every accepted dealer, Step 7 uses the public opening if its certificate is an opening; otherwise it uses the retained opening matching the accepted commitment.

Protocol description. Following the high-level idea mentioned in Section 2.4 and given the above ingredients, we formally depict our field-element DKG protocol in Fig. 16. Informally, it executes as follows.

Initially at **step 0**, all parties execute our group-element DKG protocol to enable subsequent one-round common coins, serving as the setup phase of SBC and election protocols.

From **step 1** to **step 3**, each node interacts with all other nodes and tries to form a publicly verifiable transcript of secret sharing. In particular, a node P_i commits to a secret polynomial f_i using Pedersen commitment and sends a piece of commitment $\mathsf{cm}_{i,j}$ along with its opening to each P_j (step 1). When receiving a valid opening for a commitment $\mathsf{cm}_{j,i}$ from P_j, P_i signs $\mu_{\mathsf{ack}}(j, i, \mathsf{cm}_{j,i})$ and retains the opening (step 2). Finally, a valid transcript consists of all commitments, signatures for the commitments, and valid openings without signatures. Such a transcript ensures that all nodes can receive consistent shares, since either the receiver has confirmed via its signature, or the share is contained in the transcript. Then, every node disperses the transcript to the whole network using SBC (step 3).

At **step 4**, 2κ indices are randomly sampled from $[n]$. Since nodes that remain honest through Step 3 have erased their dealing polynomials f_i, f_i' while retaining their received shares (including $f_i(i), f_i'(i)$), subsequently corrupting a selected node does not reveal its erased dealing polynomials. After the election, all nodes retrieve the transcripts dispersed by the selected nodes within the delivery phase of SBC, at **step 5**.

From **step 6** to **step 7**, each node verifies the received transcripts, and obtains its secret key share by aggregating the secret shares from the valid transcripts; then computes its public key share, and multicasts it. Finally, at **step 8**, all public key shares were already received or can be derived by interpolating the received valid public key shares.

7.3 The Analysis

Efficiency analysis. The group-element DKG in Step 0 has the communication bound $O(\lambda\kappa n^2 \log n)$ and an $O(\kappa \log n)$ round deadline from Section 5. In Steps 3 and 5, SBC handles inputs of length $O(\lambda n)$ and delivers at most 2κ of them. Lemma 15 therefore gives expected communication $O(\lambda\kappa n^2 \log n)$ for these two phases. The other steps incur $O(\lambda n^2)$ communication and $O(1)$ rounds. By linearity of expectation, the overall expected communication cost of the field-element DKG in Fig. 16 is

$$O(\lambda\kappa n^2 \log n).$$

Each $P_i \in \mathcal{P}$ performs the following steps:

Step 0: Invoke $\mathsf{RDKG}[\mathsf{sid}|\mathtt{rdkg}]$, and $\mathsf{Coin.ComSetup}[\mathsf{sid}|\mathtt{coin}]$ for common coin

Step 1: Sample t-degree polynomials: f_i, f_i' over $\mathbb{F}[X]$; commit shares: $\mathsf{cm}_{i,j} = g^{f_i(j)} h^{f_i'(j)}$

send $(\mathsf{sid}, \textsc{Share}, \mathsf{cm}_{i,j}, f_i(j), f_i'(j))$ to P_j, for each $j \in [n]$

Step 2: // upon receiving shares, return the signed commitments

receive $(\mathsf{sid}, \textsc{Share}, \mathsf{cm}_{j,i}, f_j(i), f_j'(i))$ from every $P_j \in \mathcal{P}$

for every $j \in [n]$, **if** $\mathsf{cm}_{j,i} = g^{f_j(i)} \cdot h^{f_j'(i)}$ **then**

$\mathsf{DS.Sign}(\mathsf{DS}.vk_i, \mathsf{DS}.sk_i, \mu_{\mathsf{ack}}(j, i, \mathsf{cm}_{j,i})) \to \sigma_{j,i}$; **send** $(\mathsf{sid}, \textsc{Sign}, \sigma_{j,i})$ to P_j

Step 3: // prepare and propose the PVSS transcript

receive $(\mathsf{sid}, \textsc{Sign}, \sigma_{i,j})$ from every $P_j \in \mathcal{P}$; Initialize a vector Trans_i

for every $j \in [n]$, **if** $\mathsf{DS.Vrfy}(\mathsf{DS}.vk_j, \mu_{\mathsf{ack}}(i, j, \mathsf{cm}_{i,j}), \sigma_{i,j}) = 1$ **then** $\mathsf{cert}_{i,j} = \sigma_{i,j}$

else $\mathsf{cert}_{i,j} = (f_i(j), f_i'(j))$

Invoke $\mathsf{SBC.propose}[\mathsf{sid}|\mathtt{sbc}]$ with the input $\mathsf{Trans}_i = ((\mathsf{cm}_{i,j}, \mathsf{cert}_{i,j}))_{j \in [n]}$

wait until it terminates. **erase** f_i, f_i' except $f_i(i), f_i'(i)$

Step 4: Invoke $\mathsf{Election}^{[n]^{2\kappa}}[\mathsf{sid}|\mathtt{ele}] \to \mathbb{S}$// elect a random subset of 2κ indices

Step 5: // deliver the PVSS transcripts of the chosen subset through SBC.deliver

Invoke $\mathsf{SBC.deliver}[\mathsf{sid}|\mathtt{sbc}]$ with the subset $\mathbb{S}$, and obtain the output $\{\mathsf{Trans}_z\}_{z \in \mathbb{S}}$

Step 6: Initialize $\mathsf{Qual} = \emptyset$; $\mathsf{pvk} \leftarrow \mathsf{PC.VkGen}(n, t, \mathbb{Z}_p)$// verify delivered transcripts

for every $z \in \mathbb{S}$, parse $\mathsf{Trans}_z = ((\mathsf{cm}_{z,j}, \mathsf{cert}_{z,j}))_{j \in [n]}$

if $\mathsf{PC.Vrfy}(\mathsf{pvk}, (\mathsf{cm}_{z,j})_{j \in [n]}) = 1$ and $\forall j \in [n]$, $\mathsf{CertVrfy}(z, j, \mathsf{cm}_{z,j}, \mathsf{cert}_{z,j}) = 1$, **then** $\mathsf{Qual} = \mathsf{Qual} \cup \{z\}$

Step 7: $sk_i \leftarrow \sum_{z \in \mathsf{Qual}} f_z(i), pk_i = g^{sk_i}, r_i \leftarrow \sum_{z \in \mathsf{Qual}} f_z'(i), \mathsf{cm}_i = \prod_{z \in \mathsf{Qual}} \mathsf{cm}_{z,i}$

$\pi_i \leftarrow \mathsf{NIZK.Prove}[\mathsf{sid}, i]((\mathsf{cm}_i, pk_i), (sk_i, r_i))$// prove pk_i is honestly generated

multicast (pk_i, π_i) and wait for $(pk_j', \pi_j)_{j \in [n]}$ until the next round starts

Step 8: Initialize $\mathsf{Valid} = \emptyset$// extract the public key

for $j \in [n], \mathsf{cm}_j = \prod_{z \in \mathsf{Qual}} \mathsf{cm}_{z,j}$, **if** $\mathsf{NIZK.Vrfy}[\mathsf{sid}, j]((\mathsf{cm}_j, pk_j'), \pi_j) = 1$, **then** $\mathsf{Valid} = \mathsf{Valid} \cup \{j\}$

Let $pk_j = pk_j'$ for all $j \in \mathsf{Valid}$. Let $\mathsf{Valid}' \subseteq \mathsf{Valid}$ include the first $t+1$ indices in Valid.

// Let $\{\lambda_j(x)\}_{j \in \mathsf{Valid}'}$ be Lagrange basis over Valid'

Interpolate $\{pk_j\}_{j \in \mathsf{Valid}'} : pk = \prod_{j \in \mathsf{Valid}'} pk_j^{\lambda_j(0)}$, and $pk_\tau = \prod_{j \in \mathsf{Valid}'} pk_j^{\lambda_j(\tau)}, \forall \tau \notin \mathsf{Valid}$

output $(pk, pk_1, \ldots, pk_n, sk_i)$

Fig. 16. Field-element DKG. Full secrecy is static; boxed erasures support adaptive oracle-aided simulatability.

For the deadline, let N_{tot} bound all relevant BA and MVBA invocations across the executions under consideration. Under $\log N_{\mathsf{tot}} = O(\kappa)$, the joint tail bound of Lemma 2 permits $O(\kappa)$ rounds for each randomized consensus stage, giving an overall $O(\kappa \log n)$ deadline except with negligible probability. In particular, Q field-DKG executions use $O(Q\kappa)$ BA calls in their SBC phases. Without the logarithmic parameter condition, retain $\kappa + \log N_{\mathsf{tot}}$ in the per-stage deadline. These deadline statements do not assert a worst-case communication bound for SBC or field-element DKG.

Security analysis. Regarding security, we establish the following result.

Theorem 3. *Assume $2\kappa \leq n$, the transparent setup and deadline convention of § 3, and the stated security of the consensus and common-coin components, unforgeability of* DS*, and computational binding of the Pedersen commitment (under discrete-logarithm hardness for the sampled generators). In the classical ROM, using the share proof of Lemma 18, the DKG protocol in Fig. 16 satisfies robustness and oracle-aided simulatability against strongly adaptive adversaries who can corrupt up to $t < n/2$ nodes, using the prescribed memory erasures. In addition, it achieves full secrecy against static adversaries.*

High-level intuition. First, with overwhelming probability there exists an index $i^* \in \mathbb{S}$ s.t. P_{i^*} remains honest before Step 4. Then, Trans_{i^*} must be valid, which means Qual is non-empty. On the other hand, ensured by the unforgeability of the underlying signature, and the binding property of Pedersen commitment, for every $z \in \mathsf{Qual}$, the honest nodes have valid shares from P_z. Therefore, all honest nodes can have consistent secret key shares and public key shares at step 7. Ensured by the completeness of NIZK, all public key shares from honest nodes will be computed with valid proofs and delivered to every node by the end of step 7. On the other hand, Lemma 18 lets a reduction extract an opening from a fresh inconsistent share proof with non-negligible probability, contradicting Pedersen binding. Exact copies of simulated proofs retain their prescribed share. Thus all accepted public key shares are consistent, except with negligible probability. It follows that our DKG satisfies robustness.

The proof of full secrecy is similar to the proof in [70]. In particular, the simulator will randomly select one honest node P_{i^*} and generate its PVSS transcript without knowing the committed polynomials, which can be done since the signatures of other honest nodes can be generated by the simulator. At step 4, the simulator uses the rejection sampler below to program the election oracle so that i^* is elected. Averaged over the hidden uniform choice of $i^* \in \mathbb{H}$, the accepted subset is uniform conditioned on intersecting $\mathbb{H}$, and the sampler's bounded running time adds only negligible failure probability. Then, after the set Qual is decided, the simulator recovers the public key shares of all malicious nodes and then interpolates the public key shares of honest nodes based on the given public key, which ensures the generated public key is equal to the given public key. Regarding the oracle-aided algebraic simulatability, it largely follows the proof for GJKR [56] provided in [17].

Formal proof. To facilitate our proof, we first restate the following fact about the low-degree testing that was previously discussed in [39, 70].

Lemma 19. *Let $(C_1, \ldots, C_n)$ be n group elements in $\mathbb{G}$, for which there exist t-degree polynomials f and $f' \in \mathbb{F}[X]$ s.t. $C_i = g^{f(i)}h^{f'(i)}$ for $i \in [n]$. Then, for any subset $\mathbb{I} \subset [n]$ with $|\mathbb{I}| = t+1$ and $(a_i, b_i)_{i\in\mathbb{I}}$ s.t. $C_i = g^{a_i}h^{b_i}$ for $i \in \mathbb{I}$, interpolate two t-degree polynomials a and b such that $a(i) = a_i$ and $b(i) = b_i$. It follows that $C_i = g^{a(i)}h^{b(i)}$ for all $i \in [n]$.*

Proof. Let $\{\lambda_j(X)\}_{j\in\mathbb{I}}$ be the Lagrange basis on $\mathbb{I}$. For every $i \in [n]$, interpolation in the exponent gives

$$C_i = \prod_{j\in\mathbb{I}} C_j^{\lambda_j(i)} = g^{\sum_{j\in\mathbb{I}} \lambda_j(i)a_j} h^{\sum_{j\in\mathbb{I}} \lambda_j(i)b_j} = g^{a(i)}h^{b(i)}.$$

The first equality follows by interpolating f and f'. It does not require $f = a$ or $f' = b$, which does not follow from equality of Pedersen commitments.

We prove the robustness in Lemma 20, the full secrecy in Lemma 22, and the oracle-aided algebraic simulatability in Lemma 21.

Lemma 20. *The DKG protocol in Fig. 16 satisfies robustness.*

Proof. We first argue that all participants P_i's will output some public key and public key shares at the end of the protocol execution. By the end of **Step 7**, for every $j \in [n]$, every honest P_i either (i) receives the same valid (pk_j, π_j) from P_j, or (ii) receives invalid (pk_j, π_j) or nothing from P_j. Notably, the former case happens for every honest P_j, since the properties of SBC, $\mathsf{Election}$ and low-degree test ensure: by the end of **Step 6**, all honest nodes hold the common set of valid secret sharing transcripts $\{\mathsf{Trans}_z\}_{z\in\mathsf{Qual}}$ for nonempty $\mathsf{Qual} \subseteq \mathbb{S}$, so all of them can multicast valid (pk_j, π_j) messages consistent to $\{\mathsf{Trans}_z\}_{z\in\mathsf{Qual}}$ during the **Step 7** for the completeness of NIZK generating π_j. Following that, when entering **Step 8**, all honest nodes have already received at least $n - t$ valid (pk_j, π_j) messages from distinct P_j, making them always interpolate some public key and public key shares.

Then, we argue that the output public key and public key shares are consistent and agreed, *i.e.*, there exists a t-degree polynomial $f \in \mathbb{Z}_p[X]$, s.t., all honest nodes output $pk = g^{f(0)}$ and $pk_i = g^{f(i)}$ for $i \in [n]$. Following **Step 1-3**, we know that honest nodes can always generate valid transcripts. Let $\mathbb{H}'$ be the set of all so-far-honest nodes before **Step 4**. Step 4 samples $2\kappa \le n$ distinct indices uniformly. In the ideal election, conditional on the pre-election honest set, its failure probability is

$$\Pr[\mathbb{S} \cap \mathbb{H}' = \emptyset] = \frac{\binom{n-|\mathbb{H}'|}{2\kappa}}{\binom{n}{2\kappa}} \le (t/n)^{2\kappa} < 2^{-2\kappa},$$

where the numerator is zero if fewer than 2κ parties lie outside $\mathbb{H}'$. The real election adds its negligible security error. By the validity of SBC, for every $z \in \mathbb{S} \cap \mathbb{H}'$, the transcript Trans_z delivered by SBC is exactly the transcript provided by P_z in **Step 3**. Thus, $\{\mathsf{Trans}_z\}_{z\in\mathbb{S}}$ contains honestly generated transcripts (with overwhelming probability), which means Qual is not empty.

According to **Step 7**, an honest P_i calculates sk_i as $\sum_{z\in\mathsf{Qual}} f_z(i)$. For each $z \in \mathsf{Qual}$, **Step 6** accepts $\mathsf{cert}_{z,i}$ only if it is a valid opening of $\mathsf{cm}_{z,i}$ or a signature under $\mathsf{DS}.vk_i$ on $\mu_{\mathsf{ack}}(z, i, \mathsf{cm}_{z,i})$. In the first case, the opening is available in the delivered transcript. In the second case, unforgeability implies, except with negligible probability, that P_i signed this exact acknowledgement. By **Step 2** and the retention rule, P_i therefore holds the matching opening from dealer P_z in the current session. The role, session, dealer, recipient, and commitment fields prevent signatures on any different context from satisfying this check. Hence every honest P_i has an opening for every $\mathsf{cm}_{z,i}$ with $z \in \mathsf{Qual}$. Therefore, all honest P_i can calculate sk_i, pk_i, r_i, and π_i, as per **Step 7**. Then, for the completeness of NIZK for proving public key share, $(pk'_j, \pi_j)_{j\in[n]}$ received at the end of **Step 7** contain the public key shares of all honest nodes. Since honestly generated public key shares come with a valid proof, Valid contains at least $t + 1$ indices. According to **Step 8**, the other public key shares and the public key are obtained by interpolating $\{pk_j\}_{j\in\mathsf{Valid}'}$, it follows that pk, $\{pk_j\}_{j\notin\mathsf{Valid}}$, and $\{pk_j\}_{j\in\mathsf{Valid}'}$ are consistent.

It remains to show that all honest nodes terminate with the same public key and public key shares.

In our protocol, as the corrupted nodes may not send the same public key shares to all honest nodes, different honest nodes may use different subsets of valid public key shares as Valid' to interpolate the public key and other public key shares. In the following, we show that for any set of valid public key shares, $\{pk_j\}_{j\in\mathsf{Valid}}$, there exists a t-degree polynomial f, such that $pk_j = g^{f(j)}$. If such a polynomial f exists, then all honest nodes can obtain the same public key and public key shares, as they all receive the same $t + 1$ public key shares from the forever honest nodes, which already determine the polynomial f.

We make the extraction argument explicit. Condition on the preceding transcript agreement and low-degree-test guarantees. Fix $t + 1$ forever-honest parties I. A reduction simulating these parties knows their aggregate openings (sk_i, r_i) of cm_i, for $i \in I$. Interpolate degree-at-most-t polynomials a, b through these openings. Lemma 19 implies

$$\mathsf{cm}_j = g^{a(j)}h^{b(j)}, \qquad j \in [n].$$

Any other honest party's retained opening has first coordinate $a(j)$, except with negligible probability: otherwise the reduction already has two openings of the same commitment with different first coordinates, violating Pedersen binding.

Suppose, with non-negligible probability, some honest recipient accepts an adversarial share $pk'_j \neq g^{a(j)}$. The reduction selects such a message and outputs its statement–proof pair after simulating the protocol, using the proof simulator of Lemma 18 for honest Step 7 proofs. This replacement preserves the view, including subsequent corruption replies. The selected pair is fresh: an exact simulated pair with the same (sid, j) contains the prescribed share $g^{a(j)}$. Lemma 18 therefore gives a polynomial-time rewinding reduction that, with non-negligible probability, obtains (sk'_j, r'_j) satisfying

$$\mathsf{cm}_j = g^{sk'_j} h^{r'_j}, \qquad pk'_j = g^{sk'_j}.$$

Since $pk'_j \neq g^{a(j)}$, we have $sk'_j \neq a(j)$. Together with the reference opening $(a(j), b(j))$, this breaks binding; explicitly,

$$\log_g h = \frac{sk'_j - a(j)}{b(j) - r'_j}.$$

The denominator is nonzero, since otherwise the two commitment equations imply $sk'_j = a(j)$. The reduction retains the initial run's reference opening when it forks; it need not extract every accepted proof in one execution. Applying this argument to the polynomially many candidate messages shows that all accepted shares satisfy $pk_j = g^{a(j)}$, except with negligible probability. Consequently, any $t+1$ accepted shares interpolate the same $g^{a(0)}$ and the same public key shares, completing the agreement and robustness argument. □

Lemma 21. *The DKG protocol in Fig. 16 satisfies $n(t+1)$-oracle-aided algebraic simulatability against adaptive adversaries under the assumptions of Theorem 3 and the prescribed memory erasures.*

Proof. We prove this by constructing a PPT simulator $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG,ada}}$ which simulates an execution of the DKG involving the adversary $\mathcal{A}$. For clarity, we write down the pseudo-code of $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG,ada}}$ in Fig. 17.

Let $K = n(t+1)$. The simulator is given uniformly and independently sampled group elements

$$(D_{i,\tau})_{i\in[n],\,\tau\in[0,t]} \leftarrow\!\!\$\ \mathbb{G}^K$$

and can access the discrete logarithm oracle $\mathsf{DL}_g(\cdot)$ for at most $K-1$ times. For ease of presentation, write

$$D_{i,\tau} = g^{d_{i,\tau}}$$

and define

$$f_i(X) := \sum_{\tau=0}^{t} d_{i,\tau} X^\tau.$$

Note that the simulator does not know the coefficients $d_{i,\tau}$. Since $(D_{i,\tau})_{\tau\in[0,t]}$ are uniformly and independently sampled from $\mathbb{G}$, the coefficients $(d_{i,\tau})_{\tau\in[0,t]}$ are uniformly and independently sampled from $\mathbb{Z}_p$. Hence, $f_i(X)$ is distributed identically to the random degree-t polynomial sampled by an honest dealer in the real protocol.

For every $i, j \in [n]$, the simulator can compute

$$A_{i,j} := \prod_{\tau=0}^{t} D_{i,\tau}^{j^\tau} = g^{f_i(j)}$$

without knowing $f_i(j)$. Whenever $f_i(j)$ has to be revealed to the adversary, the simulator queries $\mathsf{DL}_g(A_{i,j})$ and obtains $f_i(j)$. If P_i is corrupted before completing the polynomial erasure in **Step 3**, the simulator makes sufficiently many additional linearly independent queries to recover the entire polynomial f_i. After the erasure, the simulator only needs to recover those evaluations that remain in the corrupted party's state.

We show that, at the view of $\mathcal{A}$, the execution simulated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG,ada}}$ is indistinguishable from a real execution through the following hybrid executions.

Hybrid 0: The simulator Sim_0 acts on behalf of all so-far-honest participants by honestly following the protocol specification. When the adversary $\mathcal{A}$ corrupts a party, the simulator returns the current internal state of that party to $\mathcal{A}$.

Hybrid 1: The simulator Sim_1 is almost identical to Sim_0, except that:

$\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG},\mathsf{ada}}\left((D_{i,\tau})_{i\in[n],\,\tau\in[0,t]}\right)$

Prepare: Let $\mathbb{H}\subseteq[n]$ be the set of all so-far-honest nodes and let $\mathbb{C}:=[n]\setminus\mathbb{H}$.
Let $\mathbb{B}:=\mathbb{C}$, where $\mathbb{B}$ contains nodes corrupted before completing the Step 3 erasure.
For $i\in[n]$ and $x\in\mathbb{Z}_p$, define $A_i(x):=\prod_{\tau=0}^{t}D_{i,\tau}^{x^\tau}$.

Step 0: Honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$.
If $\mathcal{A}$ corrupts P_j, return its current internal state and update $\mathbb{H},\mathbb{C},\mathbb{B}$ accordingly.

Step 1: for $i\in\mathbb{H}$:
sample a random degree-t polynomial $f'_i(X)\leftarrow\!\!\$\ \mathbb{Z}_p[X]$.
for $j\in[n]$:
compute $A_{i,j}:=A_i(j)=\prod_{\tau=0}^{t}D_{i,\tau}^{j^\tau}$ and $\mathsf{cm}_{i,j}:=A_{i,j}h^{f'_i(j)}$.
if $j\in\mathbb{C}$: **query** $\mathsf{DL}_g(A_{i,j})$ and obtain $a_{i,j}$ such that $A_{i,j}=g^{a_{i,j}}$; **send** $(\mathsf{sid},\textsc{Share},\mathsf{cm}_{i,j},a_{i,j},f'_i(j))$ to P_j.
else : simulate the confidential delivery of $(\mathsf{sid},\textsc{Share},\mathsf{cm}_{i,j},f_i(j),f'_i(j))$ to P_j,
where $f_i(j)$ is maintained symbolically as the discrete logarithm of $A_{i,j}$.

Handling adaptive corruptions: Whenever $\mathcal{A}$ corrupts P_j, update $\mathbb{C}:=\mathbb{C}\cup\{j\}$ and $\mathbb{H}:=\mathbb{H}\setminus\{j\}$.
For every challenge-embedded dealer P_i whose polynomial has not been reconstructed, **query** $\mathsf{DL}_g(A_i(j))$
if this evaluation has not been queried before, and obtain $f_i(j)$.
if P_j has not completed the Step 3 erasure :
set $\mathbb{B}:=\mathbb{B}\cup\{j\}$.
if P_j previously behaved as an honest dealer : make additional queries $\mathsf{DL}_g(A_j(x))$ at fresh distinct points x until $t+1$ linearly independent evaluations of $f_j(X)$ are known, and interpolate $f_j(X)$.
return the reconstructed and otherwise honestly generated internal state of P_j to $\mathcal{A}$.

Step 2: Execute this step on behalf of every $P_j\in\mathbb{H}$.
For every commitment $\mathsf{cm}_{i,j}$ generated on behalf of an honest dealer P_i,
generate $\sigma_{i,j}\leftarrow\mathsf{DS.Sign}(\mathsf{DS}.vk_j,\mathsf{DS}.sk_j,\mu_{\mathsf{ack}}(i,j,\mathsf{cm}_{i,j}))$.
For shares sent by $\mathcal{A}$, follow the verification rule of the real protocol.

Step 3: For every $P_i\in\mathbb{H}$, construct Trans_i according to the real protocol.
for $j\in[n]$: **if** $\mathsf{DS.Vrfy}(\mathsf{DS}.vk_j,\mu_{\mathsf{ack}}(i,j,\mathsf{cm}_{i,j}),\sigma_{i,j})=1$: $\mathsf{cert}_{i,j}:=\sigma_{i,j}$; **else** : $\mathsf{cert}_{i,j}:=(f_i(j),f'_i(j))$.
Invoke $\mathsf{SBC.propose}[\mathsf{sid}|\mathsf{sbc}]$ with input $\mathsf{Trans}_i=((\mathsf{cm}_{i,j},\mathsf{cert}_{i,j}))_{j\in[n]}$.
After $\mathsf{SBC.propose}$ terminates, erase f_i,f'_i except for $f_i(i),f'_i(i)$.
Let $\mathbb{H}_{\mathsf{era}}:=[n]\setminus\mathbb{B}$.

Step 4: Let $\mathsf{UTS}.\sigma$ be the unique threshold signature under rpk for sid.
if $(\mathsf{sid},\mathsf{UTS}.\sigma)$ has already been queried to RO : **abort**.
Otherwise, uniformly sample $\mathbb{S}\leftarrow\!\!\$\ \{S\subseteq[n]:|S|=2\kappa\wedge S\cap\mathbb{H}_{\mathsf{era}}\neq\emptyset\}$.
Program $\mathsf{RO}(\mathsf{sid},\mathsf{UTS}.\sigma)$ so that $\mathsf{Election}[n]_{2\kappa}[\mathsf{sid}|\mathsf{ele}]$ outputs $\mathbb{S}$.

Step 5: Honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$.

Step 6: Honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$ and obtain Qual. Let $\mathbb{Q}:=\mathsf{Qual}\cap\mathbb{H}_{\mathsf{era}}$.

Step 7: for $i\in\mathbb{H}$:
compute $A_i:=\prod_{z\in\mathbb{Q}}A_z(i)=\prod_{z\in\mathbb{Q}}\prod_{\tau=0}^{t}D_{z,\tau}^{i^\tau}$, $\bar{sk}_i:=\sum_{z\in\mathsf{Qual}\setminus\mathbb{Q}}f_z(i)$, and $pk_i:=A_ig^{\bar{sk}_i}$.
simulate $\pi_i\leftarrow\mathsf{NIZK.SimProve}[\mathsf{sid},i](\mathsf{cm}_i,pk_i)$ and multicast (pk_i,π_i) on behalf of P_i.

Step 8: Honestly execute the public-key interpolation step on behalf of all $P_i\in\mathbb{H}$.
Choose any $I\subseteq\mathbb{H}$ with $|I|=t+1$, let $\{\bar{\lambda}_i(0)\}_{i\in I}$ be the Lagrange basis over I, and compute $\bar{sk}:=\sum_{i\in I}\bar{\lambda}_i(0)\bar{sk}_i$.

Finalize DLog queries: Let $\mathbb{C}$ denote the final corrupted set and choose distinct nonzero $x_1,\ldots,x_t\in\mathbb{Z}_p$
such that every index in $\mathbb{C}$ belongs to $\{x_1,\ldots,x_t\}$.
for $i\in\mathbb{H}_{\mathsf{era}}$: **query** $\mathsf{DL}_g(A_i(x_\ell))$ for every $\ell\in[t]$ whose corresponding evaluation has not been queried before.
for $i\in\mathbb{B}$:
if $<t+1$ linearly independent evaluation queries have been made for block i, **query** $\mathsf{DL}_g(A_i(x))$
at fresh distinct points x until there are $t+1$ such queries.
Choose any $i^\star\in\mathbb{Q}$. **for** $i\in\mathbb{H}_{\mathsf{era}}\setminus\{i^\star\}$: **query** $\mathsf{DL}_g(D_{i,0})$.

Output: Let $\alpha_{i,0}:=1$ for $i\in\mathbb{Q}$, and let $\alpha_{i,\tau}:=0$ for every other (i,τ).
output $\left(\bar{sk},(\alpha_{i,\tau})_{i\in[n],\,\tau\in[0,t]}\right)$ as the algebraic description of pk, where $pk=g^{\bar{sk}}\prod_{i\in[n]}\prod_{\tau=0}^{t}D_{i,\tau}^{\alpha_{i,\tau}}=g^{\bar{sk}}\prod_{i\in\mathbb{Q}}D_{i,0}$.

Fig. 17. The simulator for the oracle-aided algebraic simulatability.

- At **Step 7**, it generates the proof π_i for each so-far-honest P_i by using the simulated prover algorithm $\mathsf{NIZK.SimProve}$.

Hybrid 2: The simulator Sim_2 is almost identical to Sim_1, except that:

- Before **Step 4**, it uses the secret key shares rsk_i (w.r.t. RDKG) of honest nodes to create the unique threshold signature $\mathsf{UTS}.\sigma$ for sid. It checks whether $(\mathsf{sid}, \mathsf{UTS}.\sigma)$ has been queried by $\mathcal{A}$ to the random oracle. If it has been queried, it aborts. Otherwise, it samples the election outcome according to the same uniform distribution as in the real execution and programs $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma)$ accordingly.

Hybrid 3: Let $\mathbb{B} \subseteq [n]$ denote the set of nodes that are corrupted before completing the polynomial erasure at the end of **Step 3**, and let

$$\mathbb{H}_{\mathsf{era}} := [n] \setminus \mathbb{B}.$$

The simulator Sim_3 is almost identical to Sim_2, except that:

- At **Step 4**, instead of sampling a uniformly random subset of 2κ nodes, it samples $\mathbb{S}$ uniformly from

$$\{S \subseteq [n] : |S| = 2\kappa \ \wedge \ S \cap \mathbb{H}_{\mathsf{era}} \neq \emptyset\}$$

  and programs the random oracle so that $\mathbb{S}$ is the output of $\mathsf{Election}$.

Hybrid 4: The simulator Sim_4 is almost identical to Sim_3, except that every party P_i that is honest when acting as a dealer uses the challenge elements $(D_{i,0}, \ldots, D_{i,t})$ to generate its commitments. In particular, it computes

$$A_{i,j} = \prod_{\tau=0}^{t} D_{i,\tau}^{j^\tau} = g^{f_i(j)}$$

and samples the blinding polynomial f'_i exactly as in the real protocol. It then sets

$$\mathsf{cm}_{i,j} = A_{i,j} h^{f'_i(j)}.$$

Whenever $f_i(j)$ becomes visible to $\mathcal{A}$, the simulator obtains it by querying $\mathsf{DL}_g(A_{i,j})$.

We now argue indistinguishability between the hybrids. The execution in **Hybrid 0** is identical to the real execution. Due to the zero-knowledge property of NIZK, **Hybrid 0** and **Hybrid 1** are computationally indistinguishable.

Regarding **Hybrid 1** and **Hybrid 2**, they are identical unless $\mathcal{A}$ queries the random oracle at $(\mathsf{sid}, \mathsf{UTS}.\sigma)$ before the simulator programs this point. Due to the security of the underlying unique threshold signature scheme, this event occurs only with negligible probability. Thus, **Hybrid 1** and **Hybrid 2** are computationally indistinguishable.

We next compare **Hybrid 2** and **Hybrid 3**. Conditioned on any transcript immediately before **Step 4**, the only difference between them is the distribution of the election. In **Hybrid 2**, $\mathbb{S}$ is a uniformly random subset of 2κ nodes, while in **Hybrid 3** it is sampled from the same distribution conditioned on $\mathbb{S} \cap \mathbb{H}_{\mathsf{era}} \neq \emptyset$. Hence, their statistical distance is exactly the probability that a uniformly sampled election contains only nodes in $\mathbb{B}$, namely

$$\frac{\binom{|\mathbb{B}|}{2\kappa}}{\binom{n}{2\kappa}} \leq \left(\frac{|\mathbb{B}|}{n}\right)^{2\kappa} \leq \left(\frac{t}{n}\right)^{2\kappa} < 2^{-2\kappa},$$

where the probability is zero if $|\mathbb{B}| < 2\kappa$. Therefore, **Hybrid 2** and **Hybrid 3** are statistically indistinguishable.

Finally, consider **Hybrid 3** and **Hybrid 4**. Since every $D_{i,\tau}$ is sampled uniformly and independently from $\mathbb{G}$, the corresponding coefficients $d_{i,\tau} = \log_g D_{i,\tau}$ are uniformly and independently distributed over $\mathbb{Z}_p$. Therefore,

$$f_i(X) = \sum_{\tau=0}^{t} d_{i,\tau} X^\tau$$

has exactly the same distribution as an honestly sampled degree-t polynomial in the real protocol. The blinding polynomial f'_i is also sampled honestly. Consequently, the commitments and shares generated on behalf of every honest dealer are identically distributed to those in **Hybrid 3**. Whenever a share becomes visible due to a corruption, the simulator obtains the corresponding scalar value through the discrete logarithm oracle, and hence returns an internal state consistent with the previously generated messages. Thus, **Hybrid 3** and **Hybrid 4** are identically distributed.

The execution in **Hybrid 4** is exactly the execution generated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG},\mathsf{ada}}$. Therefore, the simulated view and the real view are computationally indistinguishable.

We next show that the simulator provides a correct algebraic expression for the final public key.

Define

$$\mathbb{Q} := \mathsf{Qual} \cap \mathbb{H}_{\mathsf{era}}.$$

By the programming of the election, $\mathbb{S} \cap \mathbb{H}_{\mathsf{era}} \neq \emptyset$. Every node in $\mathbb{S} \cap \mathbb{H}_{\mathsf{era}}$ constructs and disperses its transcript while remaining honest through the erasure point. Hence, its transcript is valid and is included in Qual. Therefore,

$$\mathbb{Q} \neq \emptyset.$$

For every $j \in [n]$, define

$$A_j := \prod_{z \in \mathbb{Q}} \prod_{\tau=0}^{t} D_{z,\tau}^{j^\tau}$$

and

$$B_j := h^{\sum_{z \in \mathbb{Q}} f'_z(j)}.$$

Let

$$\bar{\mathsf{cm}}_j := \prod_{z \in \mathsf{Qual} \setminus \mathbb{Q}} \mathsf{cm}_{z,j}.$$

We then have

$$\mathsf{cm}_j = A_j B_j \bar{\mathsf{cm}}_j.$$

For every so-far-honest P_i, the simulator knows an opening $(\bar{sk}_i, \bar{r}_i)$ of $\bar{\mathsf{cm}}_i$, where

$$\bar{sk}_i = \sum_{z \in \mathsf{Qual} \setminus \mathbb{Q}} f_z(i),$$

and it generates

$$pk_i = A_i g^{\bar{sk}_i}.$$

Consider an adversarial Step 7 message for an index j that is corrupted when the message is generated; this includes a party corrupted after previously publishing an honest proof. The simulator has recovered the evaluations at j for every $z \in \mathbb{Q}$. Set

$$a_j := \sum_{z \in \mathbb{Q}} f_z(j), \qquad b_j := \sum_{z \in \mathbb{Q}} f'_z(j),$$

so $A_j = g^{a_j}$ and $B_j = h^{b_j}$. Interpolating the known openings at $t+1$ forever-honest parties gives a reference opening $(\bar{sk}_j, \bar{r}_j)$ of $\bar{\mathsf{cm}}_j$ by Lemma 19.

Suppose an accepted message has $pk'_j \neq A_j g^{\bar{sk}_j}$ with non-negligible probability. An exact copy of a simulated proof under the same context already has the prescribed share, so any such inconsistent pair is fresh. Applying Lemma 18 to a reduction selecting this message yields an opening (sk'_j, r'_j) of cm_j with $pk'_j = g^{sk'_j}$ with non-negligible probability. Subtracting the known contribution gives

$$(sk'_j - a_j, r'_j - b_j)$$

as a second opening of $\bar{\mathsf{cm}}_j$. Its first coordinate differs from $\bar{sk}_j$, violating Pedersen binding. This is a separate binding reduction with locally sampled, known challenge exponents, as explained after Lemma 18; the oracle-aided DKG simulator itself never invokes the extractor. Hence every accepted share, including messages from parties corrupted during Step 7, satisfies $pk_j = A_j g^{\bar{sk}_j}$ except with negligible probability. Therefore, for every $j \in \mathsf{Valid}'$,

$$pk_j = g^{\bar{sk}_j} \prod_{z \in \mathbb{Q}} \prod_{\tau=0}^{t} D_{z,\tau}^{j^\tau}.$$

Let $\{\lambda_j(0)\}_{j \in \mathsf{Valid}'}$ denote the Lagrange coefficients used to interpolate the public key. It follows that

$$\begin{aligned} pk &= \prod_{j \in \mathsf{Valid}'} pk_j^{\lambda_j(0)} \\ &= g^{\bar{sk}} \prod_{z \in \mathbb{Q}} \prod_{\tau=0}^{t} D_{z,\tau}^{\sum_{j \in \mathsf{Valid}'} \lambda_j(0) j^\tau}, \end{aligned}$$

where

$$\bar{sk} := \sum_{j \in \mathsf{Valid}'} \lambda_j(0) \bar{sk}_j.$$

Since $|\mathsf{Valid}'| = t + 1$, the Lagrange interpolation identities imply

$$\sum_{j \in \mathsf{Valid}'} \lambda_j(0) j^\tau = \begin{cases} 1, & \tau = 0, \\ 0, & 1 \le \tau \le t. \end{cases}$$

Hence,

$$pk = g^{\bar{sk}} \prod_{z \in \mathbb{Q}} D_{z,0}.$$

Thus, the simulator can output the algebraic description of pk whose coefficient corresponding to $D_{z,0}$ is 1 for $z \in \mathbb{Q}$ and whose remaining challenge coefficients are 0.

It remains to prove that the simulatability matrix is invertible.

Let

$$b := |\mathbb{B}|, \qquad h := |\mathbb{H}_{\mathsf{era}}| = n - b.$$

Let $\mathbb{C}$ denote the final set of corrupted nodes. Since $|\mathbb{C}| \le t$, choose distinct nonzero field elements

$$x_1, \ldots, x_t \in \mathbb{Z}_p$$

such that every index in $\mathbb{C}$ is contained in $\{x_1, \ldots, x_t\}$. For $x \in \mathbb{Z}_p$, define

$$\mathbf{v}(x) := (1, x, \ldots, x^t).$$

The simulator organizes its discrete-logarithm queries as follows. Whenever an evaluation that has already been queried can be reused, no additional query is made.

For every $i \in \mathbb{B}$, the simulator makes exactly $t + 1$ linearly independent evaluation queries of the form

$$\mathsf{DL}_g \left(\prod_{\tau=0}^{t} D_{i,\tau}^{x^\tau} \right).$$

If P_i was initially corrupted, these queries can be made as dummy queries. If P_i was corrupted after initially behaving honestly, the queries used to reconstruct f_i are included among these $t + 1$ queries.

For every $i \in \mathbb{H}_{\mathsf{era}}$, the simulator makes exactly t evaluation queries corresponding to the distinct points $x_1, \ldots, x_t$. These queries include all evaluations that must be revealed to finally corrupted recipients; if fewer than t such evaluations are required by the execution, the remaining queries are made as dummy queries.

Since $\mathbb{Q} \neq \emptyset$, after obtaining Qual the simulator fixes an arbitrary

$$i^\star \in \mathbb{Q}.$$

Notice that $i^\star$ is chosen only for the purpose of completing the algebraic query matrix; it is not selected before the election and does not affect the simulated execution. For every

$$i \in \mathbb{H}_{\mathsf{era}} \setminus \{i^\star\},$$

the simulator additionally makes the dummy query

$$\mathsf{DL}_g(D_{i,0}).$$

The total number of discrete-logarithm queries is therefore

$$\begin{aligned} b(t+1) + ht + (h-1) &= b(t+1) + (n-b)t + (n-b-1) \\ &= n(t+1) - 1 \\ &= K - 1. \end{aligned}$$

We finally show that the resulting $K \times K$ simulatability matrix L is invertible. Partition its K columns into n blocks, where the i-th block corresponds to

$$(D_{i,0}, D_{i,1}, \ldots, D_{i,t}).$$

For every $i \in \mathbb{B}$, the $t+1$ evaluation queries give $t+1$ rows whose restriction to the i-th block forms a $(t+1) \times (t+1)$ Vandermonde matrix. Hence, these rows have full rank $t+1$ in the i-th block.

For every $i \in \mathbb{H}_{\mathsf{era}} \setminus \{i^\star\}$, the corresponding query rows restricted to the i-th block are

$$\mathbf{v}(x_1), \ldots, \mathbf{v}(x_t), \mathbf{v}(0).$$

Since $0, x_1, \ldots, x_t$ are distinct, these rows form a $(t+1) \times (t+1)$ Vandermonde matrix and hence have full rank.

For the $i^\star$-th block, the simulator has only the t rows

$$\mathbf{v}(x_1), \ldots, \mathbf{v}(x_t),$$

which have rank t. Consequently, the $K-1$ discrete-logarithm query rows are linearly independent, and the only missing dimension is the $\mathbf{v}(0)$ direction in the $i^\star$-th block.

Recall that the algebraic coefficient row of the public key is

$$\sum_{z \in \mathbb{Q}} \mathbf{v}(0)^{(z)},$$

where $\mathbf{v}(0)^{(z)}$ denotes $\mathbf{v}(0)$ in the block corresponding to z and zeros in all other blocks. For every $z \in \mathbb{Q} \setminus \{i^\star\}$, the row $\mathbf{v}(0)^{(z)}$ already appears among the discrete-logarithm query rows through the query $\mathsf{DL}_g(D_{z,0})$. Subtracting these rows from the public-key row leaves exactly

$$\mathbf{v}(0)^{(i^\star)}.$$

Since

$$\mathbf{v}(0), \mathbf{v}(x_1), \ldots, \mathbf{v}(x_t)$$

are linearly independent, this remaining row is not contained in the span of the $K-1$ discrete-logarithm query rows.

Therefore,

$$\mathsf{rank}(L) = K = n(t+1),$$

and L is invertible over $\mathbb{Z}_p$. This completes the proof.

Lemma 22. *Under the assumptions of Theorem 3, with $2\kappa \leq n$, the DKG protocol in Fig. 16 satisfies full secrecy against static adversaries.*

Proof. We prove this by constructing a PPT simulator $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ which simulates an execution of the DKG involving an adversary $\mathcal{A}$. For clarity, we write down the pseudo-code of $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ in Fig. 18, which takes as input pk, which is generated by the default key generation algorithm, and the identities of corrupted parties denoted by $\mathbb{C} \subset [n]$. We assume w.l.o.g. that $|\mathbb{C}| = t$, and put $\mathbb{H} = [n] \setminus \mathbb{C}$, $h = |\mathbb{H}|$, and $m = 2\kappa \leq n$. The designated index i^* is sampled uniformly from $\mathbb{H}$ once and remains fixed throughout the simulation, including all rejection-sampling trials.

In the following, we argue that the execution simulated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ and the real execution are indistinguishable to $\mathcal{A}$.

Hybrid 0: The simulator Sim_0 acts on behalf of all honest participants by honestly following the protocol specification.

Hybrid 1: The simulator Sim_1 is almost identical to Sim_0, except that:

- At **step 7**, it generates the proof π_i for each $i \in \mathbb{H}$ by using the simulated prover algorithm $\mathsf{NIZK.SimProve}$.

Hybrid 2: The simulator Sim_2 is almost identical to Sim_1, except that:

- Immediately before honest parties release their election shares in **step 4**, it uses their secret key shares rsk_i (w.r.t. RDKG) to compute $\mathsf{UTS}.\sigma$ for this election instance. If $(\mathsf{sid}, \mathsf{UTS}.\sigma)$ has already been queried to RO, it aborts. Otherwise, it samples $\mathbb{S} \leftarrow\!\$ \binom{[n]}{m}$ uniformly and records $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma) := \mathbb{S}$. All later queries receive this recorded value.

Hybrid 3: The simulator Sim_3 is almost identical to Sim_2, except that:

- It samples a hidden $i^* \leftarrow\!\$ \mathbb{H}$ before **step 1**, but still executes Steps 1–3 honestly for every party. At **step 4**, it retains Hybrid 2's prior-query check and replaces uniform subset sampling by the rejection sampler in Fig. 18. Each trial samples $T \leftarrow\!\$ \binom{[n]\setminus\{i^*\}}{m-1}$ uniformly, sets $S = T \cup \{i^*\}$, and accepts S with probability $1/|S \cap \mathbb{H}|$. It uses fresh independent randomness for each trial and aborts if none of the first $m\kappa$ trials accepts. Otherwise, it programs the oracle with the first accepted subset. The index i^* is not resampled.

Hybrid 4: The simulator Sim_4 is almost identical to Sim_3, except that:

- At **step 1**, after generating $C_j = \mathsf{cm}_{i^*,j}$ for all $j \in \mathbb{C}$, P_{i^*} computes $C_0 = g^{f_{i^*}(0)} h^{f'_{i^*}(0)}$, and $\mathsf{cm}_{i^*,i} = \prod_{j \in \mathbb{C} \cup \{0\}} C_j^{\lambda_j(i)}$, where $\{\lambda_j(x)\}_{j \in \mathbb{C} \cup \{0\}}$ is the Lagrange basis over $\mathbb{C} \cup \{0\}$. At **step 3**, for each $i \in \mathbb{H}$, the simulator signs $\mu_{\mathsf{ack}}(i^*, i, C_i)$ using the signing key of recipient P_i.
- At **step 7**, for every $i \in \mathbb{H}$, pk_i is calculated as follows: First, calculate $\bar{pk}_i = g^{\sum_{z \in \mathsf{Qual} \setminus \{i^*\}} f_z(i)}$. Next, let A_0 be $g^{f_{i^*}(0)}$, and interpolate $pk_i^{(i^*)} = A_0^{\lambda_0(i)} \prod_{j \in \mathbb{C}} g^{a_j \lambda_j(i)}$, where a_j (along with some b_j) for $j \in \mathbb{C}$ is the opening for $\mathsf{cm}_{i^*,j}$ sent to P_j at **step 1**. Then, let $pk_i = \bar{pk}_i pk_i^{(i^*)}$.

Hybrid 5: The simulator Sim_5 is almost identical to Sim_4, except that:

- At **step 1**, P_{i^*} does not explicitly sample f_{i^*}. Instead, it samples $A_0 \leftarrow\!\$ \mathbb{G}$, and $\{a_j\}_{j \in \mathbb{C}} \leftarrow\!\$ \mathbb{Z}_p^t$, which implicitly decides f_{i^*}, s.t. $f_{i^*}(j) = a_j$, and $A_0 = g^{f_{i^*}(0)}$. It also does not sample f'_{i^*}. Instead, it samples $\{b_j\}_{j \in \{0\} \cup \mathbb{C}} \leftarrow\!\$ \mathbb{Z}_p^{t+1}$, and computes $C_0 = A_0 h^{b_0}$.

Hybrid 6: The simulator Sim_6 is almost identical to Sim_5, except that:

- At **step 1**, P_{i^*} does not sample A_0. C_0 is uniformly sampled from $\mathbb{G}$.
- At **step 7**, A_0 is randomly chosen.

We now compare the hybrids explicitly. **Hybrid 0** is the real execution, and **Hybrid 0** and **Hybrid 1** are computationally indistinguishable by the multi-theorem zero-knowledge property of Lemma 18. **Hybrid**

$\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}(pk, \mathbb{C})$

Prepare: define $\mathbb{H} := [n] \setminus \mathbb{C}$, sample $i^* \leftarrow\!\!\$\ \mathbb{H}$

Step 0: honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 1: honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}\setminus\{i^*\}}$;

For P_{i^*}, do the following:

for $j \in \mathbb{C}$, sample $a_j, b_j \leftarrow\!\!\$\ \mathbb{Z}_p$, and compute $C_j = g^{a_j} h^{b_j}$

Let $\{\lambda_\tau(x)\}_{\tau\in\{0\}\cup\mathbb{C}}$ be the Lagrange basis over $\{0\} \cup \mathbb{C}$.

Sample $C_0 \leftarrow\!\!\$\ \mathbb{G}$, compute $C_j = \prod_{\tau\in\{0\}\cup\mathbb{C}} C_\tau^{\lambda_\tau(j)}$ for $j \in [n] \setminus \mathbb{C}$

send $(\mathsf{sid}, \textsc{Share}, C_j, a_j, b_j)$ to P_j, for each $j \in \mathbb{C}$

Step 2: honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 3: honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}\setminus\{i^*\}}$;

For P_{i^*}, follow the protocol except that:

for every $j \in \mathbb{H}$, let $\mathsf{cert}_{i^*,j} \leftarrow \mathsf{DS.Sign}(\mathsf{DS}.vk_j, \mathsf{DS}.sk_j, \mu_{\mathsf{ack}}(i^*, j, C_j))$

Step 4: Before honest election shares are released, set $m = 2\kappa \le n$.

Compute $\mathsf{UTS}.\sigma$ under rpk for this election instance.

if $(\mathsf{sid}, \mathsf{UTS}.\sigma)$ was queried to RO, **abort**.

for $\ell \in [m\kappa]$:

$T \leftarrow\!\!\$\ \binom{[n]\setminus\{i^*\}}{m-1}$; $\quad S \leftarrow T \cup \{i^*\}$; $\quad u \leftarrow\!\!\$\ [|S \cap \mathbb{H}|]$

if $u = 1$: $\mathbb{S} \leftarrow S$; **exit the loop**.

if no trial accepted, **abort**.

Program $\mathsf{RO}(\mathsf{sid}, \mathsf{UTS}.\sigma) := \mathbb{S}$.

Step 5: honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Step 6: honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$, and obtain Qual

Step 7:

for every $j \in \mathbb{C}$, compute $pk_j^{(i^*)} = g^{a_j}$, where a_j is sent on behalf of P_{i^*} in Step 1

for every $i \in \mathbb{H}$, compute $\bar{sk}_i = \sum_{z\in\mathsf{Qual}\setminus\{i^*\}} f_z(i)$, and $\bar{pk}_i = g^{\bar{sk}_i}$.

Compute $\bar{pk} = \prod_{i\in\mathbb{H}} \bar{pk}_i^{\bar{\lambda}_i(0)}$, where $\{\bar{\lambda}_i(x)\}_{i\in\mathbb{H}}$ is the Lagrange basis over $\mathbb{H}$

Compute $pk_0^{(i^*)} = pk/\bar{pk}$

for $i \in \mathbb{H}$, compute $pk_i^{(i^*)} = \prod_{j\in\{0\}\cup\mathbb{C}} (pk_j^{(i^*)})^{\hat{\lambda}_j(i)}$, where $\{\hat{\lambda}_j(x)\}_{j\in\{0\}\cup\mathbb{C}}$ is the Lagrange basis over $\{0\} \cup \mathbb{C}$

for $i \in \mathbb{H}$, compute $pk_i = pk_i^{(i^*)} \bar{pk}_i$

simulate the proof $\pi_i \leftarrow \mathsf{NIZK.SimProve}[\mathsf{sid}, i](\mathsf{cm}_i, pk_i)$

multicast (pk_i, π_i) on behalf of P_i

Step 8: honestly execute this step on behalf of all $\{P_i\}_{i\in\mathbb{H}}$

Fig. 18. The simulator for full secrecy.

1 and **Hybrid 2** are identically distributed unless the election-oracle point was queried before programming. The assumed security of the underlying common coin makes this early-query event negligible: the simulator programs the point before honest election shares are released. At a fresh point, the uniform m-subset has exactly the real oracle-output distribution.

Election sampling. Let $N = \binom{n}{m}$ and

$$\delta = \frac{\binom{n-h}{m}}{N}, \qquad \pi = 1 - \delta,$$

where $\binom{n-h}{m} = 0$ if $n - h < m$. Thus π is the probability that a uniform m-subset intersects $\mathbb{H}$. For fixed $i \in \mathbb{H}$, a trial proposes each $S \ni i$ with probability $1/\binom{n-1}{m-1}$. Write $r_H(S) = |S \cap \mathbb{H}|$. Symmetry among the h honest indices gives

$$\sum_{\substack{|S|=m \\ i \in S}} \frac{1}{r_H(S)} = \frac{N\pi}{h}.$$

Indeed, summing the left side over $i \in \mathbb{H}$ counts each subset intersecting $\mathbb{H}$ exactly once. The per-trial acceptance probability is consequently

$$a = \frac{N\pi}{h\binom{n-1}{m-1}} = \frac{n\pi}{mh},$$

independently of i. Without the trial cap, the accepted-subset distribution for a fixed designated index is

$$\mathcal{D}_{i,\mathbb{H}}(S) = \frac{h\,\mathbf{1}_{\{i \in S\}}}{N\pi\, r_H(S)} \quad \text{for } r_H(S) > 0,$$

and it is zero otherwise. Averaging over $i^* \leftarrow\!\$\, \mathbb{H}$ yields

$$\frac{1}{h}\sum_{i \in \mathbb{H}} \mathcal{D}_{i,\mathbb{H}}(S) = \frac{1}{N\pi} \quad \text{whenever } S \cap \mathbb{H} \neq \emptyset.$$

Hence the marginal election is exactly uniform conditioned on intersecting $\mathbb{H}$, while every accepted subset contains i^*. Equivalently, conditioned on the accepted subset S, the designated index is uniform in $S \cap \mathbb{H}$.

Since $1 \le r_H(S) \le m$, every trial accepts with probability at least $1/m$. The uncapped sampler takes $1/a = mh/(n\pi) \le m$ trials in expectation, and the probability of exhausting $m\kappa$ trials is

$$\epsilon_{\mathsf{sam}} = (1-a)^{m\kappa} \le (1 - 1/m)^{m\kappa} \le e^{-\kappa}.$$

The success probability is independent of i^*, so conditioning the capped sampler on success does not bias the designated index or the accepted-subset distribution above. The cap makes the sampler polynomial time, with $O(m^2\kappa)$ index-sampling operations. Its proposals and rejected trials are internal to the simulator; only the accepted subset is written to the oracle table.

In **Hybrid 3**, the hidden i^* is independent of the adversary's view before Step 4, since it has not affected any preceding protocol action. Thus the preceding distribution calculation applies conditioned on every such view for which the oracle point is fresh. The statistical distance between **Hybrid 2** and **Hybrid 3** is at most

$$\delta + \epsilon_{\mathsf{sam}} \le (t/n)^m + e^{-\kappa} < 2^{-2\kappa} + e^{-\kappa},$$

which is negligible. This comparison uses the static honest set $\mathbb{H}$ and does not rely on the adaptive simulator's different choice of a designated contribution.

Remaining hybrids. Hybrids 4–6 keep the same designated index and exactly the same sampler, prior-query check, and trial cap. For each fixed i^*, Hybrid 4 merely computes the honest commitments and shares by interpolation, so its view agrees with Hybrid 3. Hybrid 5 samples the same polynomials through their values at the $t+1$ distinct points $\{0\} \cup \mathbb{C}$, and hence preserves the joint distribution. In Hybrid 6, perfect hiding of

the Pedersen commitment permits deferring the choice of A_0 until Step 7: the view before that step, including the election and its acceptance decision, is independent of this hidden value. Thus Hybrids 5 and 6 have the same view. In particular, these comparisons preserve the correlation between i^* and the accepted subset; they do not treat the subset as uniform for a fixed i^*.

Finally, when the simulator's input pk is independently sampled from the default key distribution, setting $A_0 = pk/\bar{pk}$ has the same conditional uniform distribution as Hybrid 6's deferred choice. Therefore Hybrid 6 has the simulator's view, and the complete simulation is computationally indistinguishable from the real execution, with the additional election-sampling distance bounded above. On every non-aborting execution, the designated honest dealer is elected and its valid transcript belongs to Qual.

Finally, we show that the public key pk' output in the simulated execution equals the simulator's input pk, except with negligible probability. For $i \in \mathbb{H}$, the simulator publishes $pk_i = pk_i^{(i^*)}\bar{pk}_i$. These shares lie on a polynomial in the exponent of degree at most t, whose value at zero is $pk_0^{(i^*)}\bar{pk} = pk$.

For a corrupt index $j \in \mathbb{C}$, the simulator knows the opening (a_j, b_j) of $\mathsf{cm}_{i^*,j}$ sent in Step 1. Interpolate the openings of $\prod_{z\in\mathsf{Qual}\setminus\{i^*\}} \mathsf{cm}_{z,i}$ held at $t+1$ honest indices to obtain $(\bar{sk}_j, \bar{r}_j)$; Lemma 19 makes this an opening at j as well. Thus

$$(a_j + \bar{sk}_j, b_j + \bar{r}_j)$$

is a known reference opening of cm_j, and the consistent share is $g^{a_j+\bar{sk}_j}$.

If an adversarial proof for a different share pk'_j were accepted with non-negligible probability, a reduction selecting it could apply Lemma 18 to extract (sk'_j, r'_j) with $pk'_j = g^{sk'_j}$. The proof is fresh because the simulation oracle only issued proofs for honest indices, and the session and index are part of the hashed statement. The extracted opening and the reference opening have different first coordinates, contradicting Pedersen binding as in Lemma 20. The reduction simulates the full environment in both runs; it uses extraction only to establish this contradiction. Therefore every accepted share lies on the same polynomial in the exponent as the honest shares. Step 8 consequently outputs $pk' = pk$. □

8 Related Work

8.1 Synchronous Common Coin Protocols

We overview the synchronous common coin constructions in the following four categories.

"Commit-and-Reveal" Coins. The most classical way to generate a common coin is via the "commit-and-reveal" paradigm [24, 59], where each node P_i commits to a random value r_i and *broadcasts* the commitment c_i, and then the network opens all commitments and obtains the common coin $r = \sum \mathsf{Open}(c_i) = \sum r_i$. For security, the underlying *broadcast* mechanism needs to ensure all nodes receive the same broadcasted value despite a potentially malicious sender, and the underlying commitment scheme should support "forced-opening", i.e., the commitment can be opened under certain conditions despite a potentially malicious committer. The candidate commitment schemes include verifiable secret sharing schemes [57], timed commitment schemes [71], and threshold encryption schemes [13, 54].

Both the commitment scheme (i.e., the commitment size and its opening mechanism) and the broadcast mechanism matter to the overall communication cost, and there are independent efforts in pushing each direction forward [13, 33, 54]. In particular, Garg et al. [54] give a commitment scheme with $O(\lambda)$-commitment size (when the size of the committed message is $O(\lambda)$) and adaptive security (as observed in [48]), under the setup of PKI and CRS. Avitabile et al. [13] give a commitment scheme with $O(\mathsf{poly}(\log n)\lambda)$-commitment size and static security, under a PKI setup. The broadcast mechanism also contributes substantially to the cost. Parallel broadcast (PBC) lets every party broadcast an ℓ-bit input so that honest parties agree on the resulting vector. Under strong adaptive corruption, the construction of [10] gives expected $O(n^2\ell + \lambda n^3 \log n)$ communication with the signature and witness sizes used here. The construction of [33] gives $O(n^2\ell + \lambda n^2 \log n)$ communication using $O(\lambda \log n)$-bit cryptographic objects, and $O(n)$ rounds. Its abstract bound becomes $O(n^2\ell + \lambda n^2)$ when those objects have $O(\lambda)$ bits. In commit-and-reveal coins, each input to PBC is a

commitment. In Table 1, we list two instantiations, where the PBC is the state-of-the-art construction in [33], while the commitment schemes are from [54] and [13], respectively.

Coins from leader-based consensus. Recent practical systems such as [21,22,38] avoid explicitly invoking PBC by adopting a *leader*-based approach. With an honest leader, the network can agree on a vector of input commitments using $O(n^2\lambda)$ communication and $O(1)$ rounds; with a dishonest leader, the network detects the failure and switches to a new leader. Since there can be up to $n/2$ malicious leaders, the first common coin is guaranteed only after $n/2$ leaders, at which point $O(n^3)$ communication and $O(n)$ rounds have been spent.

Threshold VRF-based coins. The above approaches generate coins with "true entropy." However, in the computational setting, once a secret key with "true entropy" is available, polynomially many pseudorandom values can be generated by applying a pseudorandom function. In the distributed setting, this idea can be realized using a non-interactive threshold verifiable random function (VRF) [16, 25]. Specifically, each node holds a secret share of the key sk as its private input. Then, when the network requires a new common coin, the nodes jointly evaluate the VRF to obtain $\mathsf{VRF}(sk, \mathsf{sid})$ for the session identity sid, which serves as the coin value.

This approach is highly efficient, requiring $O(\lambda n^2)$ communication and only one round, and has been adopted in practical systems such as DRand [1]. Adaptive security can be achieved provided that the underlying threshold VRF scheme is itself adaptively secure, as in the constructions of [16, 17]. However, this approach requires a setup phase to distribute secret key shares among the nodes, which can be accomplished either by a trusted dealer or via Distributed Key Generation (DKG), as we elaborate below.

DKG-based Coins. Distributed Key Generation (DKG) [28, 51, 56, 62, 68] can be viewed as a "secret version" of a common coin: instead of publicly revealing randomness, the parties jointly generate a secret value s and each holds a share s_i, so that s can serve as a cryptographic key. A common coin can be directly obtained from DKG by reconstructing s and computing $\mathsf{Hash}(s)$. In addition, once DKG is available as preprocessing, coins can also be generated efficiently ($O(n^2)$ communication and one round) using a non-interactive threshold VRF scheme [25, 27]

However, DKG remains the clear bottleneck in this approach. Most existing DKG protocols require $O(n^3)$ communication and $O(n)$ rounds [16, 46]. Shrestha et al. [70] reduce the round complexity to $O(1)$, but their protocol still requires $O(n^3)$ communication. Gurkan et al. [60] present a DKG protocol achieving near-quadratic communication $O(\lambda n^2 \log n)$ with linear round complexity $O(n)$ when instantiated with the PBC of [33]; however, it only tolerates $O(\log n)$ Byzantine parties. Feng et al. [47] propose a DKG with near-quadratic communication, but their construction assumes an initial random value after PKI setup yet before the protocol execution, and it provides only weak adaptive security.

Two recent works, Dragon-DKG [46] and Grand-DKG [16], finally achieve *subcubic* communication ($O(\lambda n^{2.5} \log n)$ and $O(\lambda n^2 \log n)$, respectively) and honest-majority resilience in the PKI model with transparent parameters. However, to achieve these communication improvements, both rely on deterministic Byzantine Agreement (BA), which by the classical lower bound of Fischer and Lynch [49] requires $O(n)$ rounds.

Other approaches. Beyond the above major categories, there are also proposals for building common coins using verifiable random functions (VRFs) [40, 58], verifiable delay functions (VDFs) [26], or proof of work [61].

The VRF-based approach, most notably Algorand [58], employs a VRF to sample a small committee, which then generates a common coin and disseminates it to the network. This approach, however, does not attain the same honest-majority resilience and is inherently not strongly adaptive, due to the nature of committee sampling. In addition, as analyzed in [16], it does not guarantee standard unpredictability.

The VDF-based approach resembles the commit-and-reveal method, but replaces the commitment with a VDF. A VDF requires a prescribed amount of time to compute but can be verified quickly. These schemes assume a broadcast channel as a black box, and thus their communication and round complexity depend on the underlying broadcast mechanism, inheriting the same issues discussed for commit-and-reveal. Moreover, VDFs require specialized hardware for efficient computation.

Finally, protocols such as RandChain [61] are built directly upon proof-of-work-based Nakamoto consensus, thereby avoiding the communication barrier of the broadcast mechanism inherent in commit-and-reveal. However, proof-of-work relies on a different type of honest-majority assumption: namely, that honest parties

control the majority of hashing power, rather than the majority of parties being honest, which may be inconsistent with various application scenarios. It also inherits the vulnerabilities of the underlying consensus protocol. Furthermore, as shown in [16], RandChain does not guarantee standard unpredictability; i.e., the random coins may be predicted well in advance by an adversary.

8.2 Asynchronous or Partially Synchronous Common Coins

The asynchronous or partially synchronous model does not assume a publicly known bound on message delivery, and thus relies on weaker assumptions than the synchronous model. Consequently, common coin protocols in the asynchronous or partially synchronous setting also imply a protocol in the synchronous setting. That said, all known asynchronous or partially synchronous protocols (except those of [27], which assume a trusted setup) only work for $n \geq 3t+1$, whereas our synchronous protocol works for any $n \geq 2t+1$, offering higher resilience. In addition, all asynchronous or partially synchronous protocols incur at least cubic communication cost, except for the recent concurrent works of [4, 5, 48]. We refer the reader to [48] for a comprehensive review of asynchronous common coins.

Comparison with [48]. The common coin protocol of [48] assumes a silent setup, in which a structured common reference string is available to all nodes, and each node must post an $O(n\lambda)$-sized public key to the bulletin board. In contrast, our protocol uses standard PKI and transparent public parameters (including a URS), where no structured reference string with a secret trapdoor is required and each node posts only a $\widetilde{O}(1)$-sized public key. Furthermore, while [48] achieves an optimal common coin in the asynchronous setting with $n \geq 3t+1$, their techniques do not extend to the synchronous setting with $n \geq 2t+1$. For example, their protocol heavily relies on a quorum certificate (i.e., a threshold signature with threshold $n-t \geq 2t+1$), which is not applicable when $n = 2t+1$.

Comparison with [4, 5]. Both works use a verifiable random function to sample $O(\kappa)$ PVSS transcripts, ensuring that at least one selected transcript is generated by an honest party and thus preserves secrecy of the resulting key. Their adaptive security therefore relies on memory erasures: without erasures, an adaptive adversary could eventually corrupt all $O(\kappa)$ selected parties and recover the information needed to compromise the final secret key. Moreover, some essential components of these constructions require $n \geq 3t+1$, preventing a direct extension to the synchronous setting with optimal resilience $n \geq 2t+1$.

8.3 Relevant Consensus Protocols

We discuss consensus protocols that can be used to implement broadcast channels in existing Distributed Key Generation (DKG) protocols. The most standard approach is to invoke a Byzantine broadcast (BB) [42] instance whenever a sender wishes to broadcast a message to the network. The validity property of BB ensures that if the sender is honest, every participant receives the message as it was broadcast. The agreement property guarantees that all receivers decide on the same message, even if the sender is malicious. Finally, the termination property requires that the protocol concludes within a bounded number of rounds.

Note that a PKI setup for digital signatures is necessary for BB and Byzantine Agreement (BA) protocols that tolerate more than $n/3$ corrupted nodes [50]. Dolev and Strong [42] introduced the first BB protocol capable of tolerating up to $n-1$ corrupted nodes. With standard digital signatures, their BB protocol incurs a communication cost of $O(n^2\ell + \lambda n^3)$ and requires $O(n)$ rounds to broadcast an ℓ-bit input to a network of n nodes. If an aggregatable signature scheme like BLS is employed, the communication cost can be reduced to $O(n^2\ell + \lambda n^2 + n^3)$. Since a classical DKG scheme invokes $O(n)$ BB instances, the total communication cost is considered $O(n^4)$, as noted in [16, 70].

A few recent works have pushed the communication complexity of BB to be nearly optimal. In particular, Momose and Ren [66] gave a BA protocol with $O(n^2\ell + \lambda n^2)$ communication cost and $O(n)$ rounds, tolerating any $t < n/2$ corrupted nodes. Their BA protocol assumes a constant-sized threshold signature scheme. However, the threshold signature scheme in their protocol is only for compressing a certificate consisting of $O(n)$ digital signatures, so we can use the recent DKG-free threshold signature schemes to instantiate,

without causing a circularity issue in DKG constructions. As we have discussed in § 3.3, under a transparent setup, we have a threshold signature scheme whose size is $O(\lambda \log n)$. Therefore, it leads to a BA protocol with $O(n^2\ell + \lambda n^2 \log n)$ communication complexity under a transparent setup. Additionally, Nayak et al. [67] gave an extension technique, which can adapt a BA for short inputs like Momose and Ren [66] for handling long inputs. Notably, combining Momose and Ren [66] and Nayak et al. [67] gives us a BA protocol with $O(n\ell + \lambda n^2 \log n)$ communication complexity and $O(n)$ rounds. Note that for $t < n/2$, there is a trivial reduction from BB to BA, where the sender just needs to multicast its input, and then the network runs a BA for reaching a consensus. Therefore, it naturally follows a BB protocol with the same complexity. If a CRS such as the q-SDH parameter is allowed, we can remove the factor of $O(\log n)$ from the communication cost.

We remark that Bacho et al. [15] also gave a BB protocol with $\widetilde{O}(n\ell + \lambda n^2)$ communication cost and $O(n)$ rounds, which does not require threshold signatures. However, their protocol only achieves static security.

Alternatively, since DKG involves $O(n)$ broadcasters, one can use a parallel Byzantine broadcast (PBC) protocol to implement the broadcast channel. For deterministic protocols, Feng et al. [46] proposed a PBC with a communication complexity of $O(n^{2.5}\ell + \lambda n^{2.5} \log n)$, while Civit et al. [33] introduced one with a complexity of $O(n^2\ell + \lambda n^2 \log n)$, under a transparent setup. However, in classical DKG schemes such as those by Pedersen [68] and Gennaro et al. [56], each node may broadcast $O(\lambda n)$ bits, meaning that advanced PBC protocols are unlikely to outperform the cubic communication complexity. On the other hand, the DKG protocol by Gurkan et al. [60] can benefit from the improved PBC protocols, achieving a reduced communication complexity of $O(\lambda n^2 \log^2 n)$.

Table 3. Comparison with existing synchronous DKG protocols.

Protocol	Resi.	Comm.Cost	Round	Field?	Adap.?
Pedersen [68]	1/2	$O(\lambda n^3 \log n)$	$O(n)$	✓	✓
GJKR [56]	1/2	$O(\lambda n^3 \log n)$	$O(n)$	✓	✓
Katz [62]	1/2	$O(\lambda n^3 \log n)$	$O(n)$	✓	✗
CGJ+ [28]	1/2	$O(\lambda n^3 \log n)$	$O(n)$	✓	✓*
GJM+ [60]	$\log n/n$	$O(\lambda n^2 \log^2 n)$	$O(n)$	✗	✗
SBKN [70]	1/2	$O(\lambda n^3 \log n)$	$O(1)$	✓	✗
GRandLine [16]	1/2	$O(\lambda n^2 \log n)$	$O(n)$	✗	✓
Dragon-DKG-I [46]	1/2	$O(\lambda n^{2.5} \log n)$	$O(n)$	✗	✓
Dragon-DKG-II [46]	1/2	$O(\lambda\kappa n^{2.5} \log n)$	$O(n)$	✓	✗
This work-I (§ 5)	1/2	$O(\lambda\kappa n^2 \log n)$	$O(\kappa \log n)$	✗	✓
This work-II (§ 7)	1/2	$O(\lambda\kappa n^2 \log n)$	$O(\kappa \log n)$	✓	✓

`Resi.`: the maximal fraction of Byzantine nodes the protocol can tolerate.
`Comm.Cost` measures the number of bits sent by all honest nodes; This work-I uses the prescribed MVBA cutoff, whereas This work-II gives an expected bound. λ and κ are the computational security parameter and the statistical security parameter, respectively.
`Round:` SBKN [70] requires $O(1)$ rounds in *expectation*, while ours can terminate within $O(\kappa \log n)$ rounds with overwhelming probability when $\log N = O(\kappa)$ for the total invocation bound in Lemma 2.
`Adap.?` asks if the protocol is adaptively secure. For This work-II, the checkmark means adaptive robustness and oracle-aided simulatability with memory erasures; full secrecy is proved against static adversaries, and $2\kappa \leq n$ is assumed. Most protocols achieve the relaxed adaptive security definition by [17], but CGJ+ (*) achieves the standard one.
`Field?` asks if the secret key is in a scalar field.

8.4 Comparison with Distributed Key Generation Protocols

Table 3 compares our two DKG protocols (one for group element and the other for field element) with existing synchronous DKG protocols. The comparison uses standard PKI and transparent public parameters (including a URS), as in § 3. The comparison assumes the state-of-the-art transparent-setup consensus protocols for implementing broadcast channels.

9 Conclusion and Future Directions

In this paper, we presented the first strongly adaptive common coin protocol under standard PKI and transparent public parameters, without a trusted dealer, that tolerates the optimal $t < n/2$ corruptions in

synchronous networks. It achieves near-quadratic communication $O(\kappa\lambda n^2 \log n)$ and $O(\kappa \log n)$ rounds, and requires no memory erasures. At its core is PCSBB, a parallel consortium-sender Byzantine broadcast that relies only on *endemic* coins within individual consortia. Applying PCSBB recursively, following our Circular Dragon methodology, yields a group-element DKG that bootstraps the common coin from scratch. Building on this coin, we further obtained a strongly adaptive PBC protocol with $O(\kappa n^2 \ell + \kappa\lambda n^2 \log n)$ communication and $O(\kappa \log n)$ rounds under standard PKI, and a field-element DKG with $O(\kappa\lambda n^2 \log n)$ communication and $O(\kappa \log n)$ rounds that is compatible with legacy threshold schemes such as BLS, Schnorr, and ElGamal.

Removing the algebraic group model. Our adaptive security analysis assumes the AGM because the adaptive security of the underlying aggregatable PVSS scheme is only established in this model [18]. Very recently, Bacho, Chen, and Loss [14] constructed adaptively secure (aggregatable) PVSS from standard assumptions. In principle, our recursive DKG can be instantiated with their scheme, removing the reliance on the AGM. The remaining gap is a one-round common coin protocol that is secure under standard assumptions and whose keys are compatible with the new APVSS, so that our recursive DKG can serve as its setup. We believe such a protocol is achievable and leave it to future work.

Removing the factor κ. Both the communication and the round complexities of our protocols carry a multiplicative factor of the statistical security parameter κ. In the common coin, this factor stems from running each randomized consensus instance inside PCSBB for a fixed deadline of $O(\kappa)$ rounds, so that all instances terminate except with probability $2^{-\kappa}$, at each of the $O(\log n)$ recursion levels. In PBC and the field-element DKG, it additionally arises from the randomly elected subset of $O(\kappa)$ senders. An interesting open question is whether this factor can be removed from both complexities, e.g., achieving $O(\lambda n^2 \log n)$ communication and $O(\log n)$ rounds, while preserving strong adaptive security and optimal resilience.

Acknowledgement

Part of this work was done when Hanwen was at USYD. Yuan's contributions to this research were made while he was at ISCAS.

References

1. drand. https://drand.love/
2. Tor shared random subsystem specification. https://spec.torproject.org/srv-spec/index.html
3. Abraham, I., Asharov, G., Patil, S., Patra, A.: Asymptotically free broadcast in constant expected time via packed VSS. In: TCC (1). Lecture Notes in Computer Science, vol. 13747, pp. 384–414. Springer (2022)
4. Abraham, I., Bacho, R., Loss, J., Stern, G.: Nearly quadratic asynchronous distributed key generation from recursive consensus. In: PODC. pp. 247–258. ACM (2026)
5. Abraham, I., Bacho, R., Stern, G.: Quadratic asynchronous DKG from plain setup. IACR Cryptol. ePrint Arch. **2026**, 1159 (2026)
6. Abraham, I., Chan, T.H., Dolev, D., Nayak, K., Pass, R., Ren, L., Shi, E.: Communication complexity of byzantine agreement, revisited. Distributed Comput. **36**(1), 3–28 (2023)
7. 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: Financial Cryptography. Lecture Notes in Computer Science, vol. 11598, pp. 320–334. Springer (2019)
8. Abraham, I., Malkhi, D., Nayak, K., Ren, L., Yin, M.: Sync hotstuff: Simple and practical synchronous state machine replication. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 106–118. IEEE (2020)
9. Abraham, I., Nayak, K., Shrestha, N.: Optimal good-case latency for rotating leader synchronous bft. Cryptology ePrint Archive (2021)
10. Abraham, I., Nayak, K., Shrestha, N.: Communication and round efficient parallel broadcast protocols. Cryptology ePrint Archive (2023)
11. Adida, B.: Helios: Web-based open-audit voting. In: USENIX Security Symposium. pp. 335–348. USENIX Association (2008)
12. Attema, T., Cramer, R., Rambaud, M.: Compressed Σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In: ASIACRYPT (4). Lecture Notes in Computer Science, vol. 13093, pp. 526–556. Springer (2021)
13. Avitabile, G., Döttling, N., Magri, B., Sakkas, C., Wohnig, S.: Signature-based witness encryption with compact ciphertext. In: ASIACRYPT (1). Lecture Notes in Computer Science, vol. 15484, pp. 3–31. Springer (2024)
14. Bacho, R., Chen, Y., Loss, J.: Adaptively secure (aggregatable) PVSS from standard assumptions. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 16801, pp. 166–197. Springer (2026)
15. Bacho, R., Collins, D., Liu-Zhang, C., Loss, J.: Network-agnostic security comes (almost) for free in DKG and MPC. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 14081, pp. 71–106. Springer (2023)
16. Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S., Papachristoudis, D.: Grandline: Adaptively secure DKG and randomness beacon with (almost) quadratic communication complexity. In: CCS. ACM (2024)
17. Bacho, R., Loss, J.: On the adaptive security of the threshold BLS signature scheme. In: CCS. pp. 193–207. ACM (2022)
18. Bacho, R., Loss, J.: Adaptively secure (aggregatable) pvss and application to distributed randomness beacons. In: CCS. ACM (2023)
19. Baird, L., Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: Threshold signatures in the multiverse. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 1454–1470. IEEE (2023)
20. Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distributed Comput. **16**(4), 249–262 (2003)
21. Bhat, A., Kate, A., Nayak, K., Shrestha, N.: Optrand: Optimistically responsive distributed random beacons. In: NDSS (2023)
22. Bhat, A., Shrestha, N., Luo, Z., Kate, A., Nayak, K.: Randpiper - reconfiguration-friendly random beacons with quadratic communication. In: CCS. pp. 3502–3524. ACM (2021)
23. Blahut, R.E.: Theory and practice of error control codes. Addison-Wesley (1983)
24. Blum, M.: Coin flipping by telephone. In: CRYPTO. pp. 11–15. U. C. Santa Barbara, Dept. of Elec. and Computer Eng., ECE Report No 82-04 (1981)
25. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 2567, pp. 31–46. Springer (2003)
26. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (1). Lecture Notes in Computer Science, vol. 10991, pp. 757–788. Springer (2018)
27. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In: PODC. pp. 123–132. ACM (2000)

28. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: CRYPTO. Lecture Notes in Computer Science, vol. 1666, pp. 98–115. Springer (1999)
29. Cascudo, I., David, B.: SCRAPE: scalable randomness attested by public entities. In: ACNS. Lecture Notes in Computer Science, vol. 10355, pp. 537–556. Springer (2017)
30. Chase, M., Lysyanskaya, A.: On signatures of knowledge. In: CRYPTO. Lecture Notes in Computer Science, vol. 4117, pp. 78–96. Springer (2006)
31. Chaum, D., Pedersen, T.P.: Wallet databases with observers. In: CRYPTO. LNCS, vol. 740, pp. 89–105. Springer (1992)
32. Choi, K., Manoj, A., Bonneau, J.: Sok: Distributed randomness beacons. In: SP. pp. 75–92. IEEE (2023)
33. Civit, P., Dzulfikar, M.A., Gilbert, S., Guerraoui, R., Komatovic, J., Vidigueira, M.: DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In: PODC. pp. 145–156. ACM (2024)
34. Civit, P., Emdin, G., Guerraoui, R.: Constant-round interactive consistency in dishonest majority, or with quadratic communication in honest majority. Cryptology ePrint Archive, Paper 2026/1146 (2026), https://eprint.iacr.org/2026/1146
35. Cohen, R., Doerner, J., Lee, E., Lysyanskaya, A., Roy, L.: An unstoppable ideal functionality for signatures and a modular analysis of the dolev-strong broadcast. Cryptology ePrint Archive, Paper 2024/1807 (2024), https://eprint.iacr.org/2024/1807
36. Collins, D., Duan, S., Loss, J., Papamanthou, C., Tsimos, G., Wang, H.: Towards optimal parallel broadcast under a dishonest majority. In: FC (2025)
37. Das, S., Camacho, P., Xiang, Z., Nieto, J., Bünz, B., Ren, L.: Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 356–370 (2023)
38. Das, S., Krishnan, V., Isaac, I.M., Ren, L.: Spurt: Scalable distributed randomness beacon with transparent setup. In: SP. pp. 2502–2517. IEEE (2022)
39. Das, S., Xiang, Z., Tomescu, A., Spiegelman, A., Pinkas, B., Ren, L.: Verifiable secret sharing simplified. In: SP. pp. 633–651. IEEE (2025)
40. Dodis, Y., Yampolskiy, A.: A verifiable random function with short proofs and keys. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 3386, pp. 416–431. Springer (2005)
41. Dolev, D., Reischuk, R.: Bounds on information exchange for byzantine agreement. In: PODC. pp. 132–140. ACM (1982)
42. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
43. Faust, S., Kohlweiss, M., Marson, G.A., Venturi, D.: On the non-malleability of the Fiat–Shamir transform. In: Progress in Cryptology – INDOCRYPT 2012. Lecture Notes in Computer Science, vol. 7668, pp. 60–79. Springer (2012), https://www.pure.ed.ac.uk/ws/files/55405394/On_the_Non_malleability.pdf
44. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: STOC. pp. 148–161. ACM (1988)
45. Feng, H., Gao, Y., Lu, Y., Tang, Q., Xu, J.: Practical asynchronous distributed key reconfiguration and its applications. Cryptology ePrint Archive (2025)
46. Feng, H., Lu, Z., Tang, Q.: Dragon: Decentralization at the cost of representation after arbitrary grouping and its applications to sub-cubic DKG and interactive consistency. In: PODC. pp. 469–479. ACM (2024)
47. Feng, H., Mai, T., Tang, Q.: Scalable and adaptively secure any-trust distributed key generation and all-hands checkpointing. In: ACM CCS (2024)
48. Feng, H., Tang, Q.: Asymptotically optimal adaptive asynchronous common coin and DKG with silent setup. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 16002, pp. 642–675. Springer (2025)
49. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Inf. Process. Lett. **14**(4), 183–186 (1982)
50. Fischer, M.J., Lynch, N.A., Merritt, M.: Easy impossibility proofs for distributed consensus problems. In: PODC. pp. 59–70. ACM (1985)
51. Fouque, P., Stern, J.: One round threshold discrete-log key generation without private channels. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 1992, pp. 300–316. Springer (2001)
52. Garay, J.A., Katz, J., Kumaresan, R., Zhou, H.: Adaptively secure broadcast, revisited. In: PODC. pp. 179–186. ACM (2011)
53. Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., Zhang, Y.: hints: Threshold signatures with silent setup. In: 2024 IEEE Symposium on Security and Privacy (SP). pp. 3034–3052. IEEE (2024)
54. Garg, S., Kolonelos, D., Policharla, G., Wang, M.: Threshold encryption with silent setup. In: CRYPTO (7). Lecture Notes in Computer Science, vol. 14926, pp. 352–386. Springer (2024)

55. Gelles, R., Lenzen, C., Loss, J., Yandamuri, S.: Nearly optimal parallel broadcast in the plain public key model. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 16001, pp. 101–134. Springer (2025)
56. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. J. Cryptol. **20**(1), 51–83 (2007)
57. Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J.B., Rabin, T., Yakoubov, S.: YOSO: you only speak once - secure MPC with stateless ephemeral roles. In: CRYPTO (2). Lecture Notes in Computer Science, vol. 12826, pp. 64–93. Springer (2021)
58. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: Scaling byzantine agreements for cryptocurrencies. In: SOSP. pp. 51–68. ACM (2017)
59. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: STOC. pp. 218–229. ACM (1987)
60. Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Aggregatable distributed key generation. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 12696, pp. 147–176. Springer (2021)
61. Han, R., Lin, H., Yu, J.: RandChain: A scalable and fair decentralised randomness beacon. Cryptology ePrint Archive, Paper 2020/1033 (2020), https://eprint.iacr.org/2020/1033
62. Katz, J.: Round-optimal, fully secure distributed key generation. In: CRYPTO (7). Lecture Notes in Computer Science, vol. 14926, pp. 285–316. Springer (2024)
63. Katz, J., Koo, C.: On expected constant-round protocols for byzantine agreement. In: CRYPTO. Lecture Notes in Computer Science, vol. 4117, pp. 445–462. Springer (2006)
64. LAMPORT, L., SHOSTAK, R., PEASE, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems **4**(3), 382–401 (1982)
65. Lindell, Y., Lysyanskaya, A., Rabin, T.: On the composition of authenticated byzantine agreement. In: STOC. pp. 514–523. ACM (2002)
66. Momose, A., Ren, L.: Optimal communication complexity of authenticated byzantine agreement. In: DISC. LIPIcs, vol. 209, pp. 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
67. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: DISC. LIPIcs, vol. 179, pp. 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
68. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: CRYPTO. Lecture Notes in Computer Science, vol. 576, pp. 129–140. Springer (1991)
69. Rabin, M.O.: Randomized byzantine generals. In: FOCS. pp. 403–409. IEEE Computer Society (1983)
70. Shrestha, N., Bhat, A., Kate, A., Nayak, K.: Synchronous distributed key generation without broadcasts. Tech. rep., Cryptology ePrint Archive (2021), https://eprint.iacr.org/2021/1635
71. Thyagarajan, S.A.K., Castagnos, G., Laguillaumie, F., Malavolta, G.: Efficient CCA timed commitments in class groups. In: CCS. pp. 2663–2684. ACM (2021)
72. Tsimos, G., Loss, J., Papamanthou, C.: Gossiping for communication-efficient broadcast. In: CRYPTO (3). Lecture Notes in Computer Science, vol. 13509, pp. 439–469. Springer (2022)
73. Zamani, M., Movahedi, M., Raykova, M.: Rapidchain: Scaling blockchain via full sharding. In: CCS. pp. 931–948. ACM (2018)