Dragon: Decentralization at the cost of Representation after Arbitrary Grouping and Its Applications to Sub-cubic DKG and Interactive Consistency

HANWEN FENG, The University of Sydney, Australia
ZHENLIANG LU, The University of Sydney, Australia
QIANG TANG, The University of Sydney, Australia

Several distributed protocols, including distributed key generation (DKG) and interactive consistency (IC), depend on $O(n)$ instances of Byzantine Broadcast or Byzantine Agreement among n nodes, resulting in $\Theta(n^3)$ communication overhead.

In this paper, we provide a new methodology of realizing such broadcasts we call DRAGON: Decentralization at the cost of Representation after Arbitrary GrOupiNg. At the core of it, we arbitrarily group nodes into small "shards" and paired with multiple new primitives we call consortium-sender (dealer) broadcast (and secret sharing). The new tools enable a shard of nodes to jointly broadcast (or securely contribute a secret) to the whole population only at the cost of one dealer (*as if* there is a representative).

With our new Dragon method, we construct the first two DKG protocols, both achieving optimal resilience, with sub-cubic total communication and computation. The first DKG generates a secret key within an Elliptic Curve group, incurring $\widetilde{O}(n^{2.5}\lambda)$ total communication and computation. The second DKG, while slightly increasing communication and computation by a factor of the statistical security parameter, generates a secret key as a field element, which makes it directly compatible with various off-the-shelf DLog-based threshold cryptographic systems. We also construct a first deterministic IC with sub-cubic communication. Along the way, we also formalize simulation-based security and proved it for publicly verifiable secret sharing (PVSS), making it possible for a modular analysis, which might be of independent interest.

1 INTRODUCTION

Many distributed protocols rely on $O(n)$ instances of Byzantine Broadcast or Byzantine Agreement among n nodes, thus incurring a $\Theta(n^3)$ communication complexity. Two notable examples are distributed key generation (DKG) and interactive consistency (IC).

Distributed key generation. DKG protocols [18, 27, 33, 34, 36, 41, 49, 53] enable a group of participants to collaboratively create a public key, while each member obtains a secret share of the secret key. This decentralized approach eliminates the need to rely on a trusted party to do the key generation process, serving as one indispensable building block in many distributed protocols, such as secure multiparty computation [24], Byzantine consensus [17, 40], threshold cryptography, and various emerging applications in blockchain (cryptocurrency wallets[35], securing proof of stake against long-range attacks via checkpointing [6], and more).

Despite its fundamental importance, existing DKG protocols involving n participants suffer from a total communication [1] cost of $\Omega(n^3\lambda)$, [2] which is prohibitive even for moderate-scale deployments, particularly when DKG is required to be continuously run in several settings, e.g., [6]. The cubic complexity in DKG arises from the common design method of collectively executing verifiable secret sharing (VSS) [23, 26, 53]. In a nutshell, among n participants where up to t could be adversarial, each participant P_i picks a t-degree polynomial f_i to define $sk^{(i)} = f_i(0)$. They then *deliver* the share $sk_j^{(i)} = f_i(j)$ to other P_j and *broadcast* a commitment, com_i, for the polynomial $f_i(X)$. Participants validate received shares and may collectively engage in a *complaint* phase, identifying the set $\mathbb{J} \in [n]$,

[1] Throughout the paper, communication cost refers to the *total* communicated bits of honest participants during one execution.
[2] The only exception is a very recent concurrent work [7]. We use very different techniques and lead to quite different trade-offs on security, functionality, and efficiency; we will explain the differences in detail at the end of Sect.2.

ensuring all transmitted secret shares are valid. The final secret share for P_i is $sk_i = \sum_{j\in\mathbb{J}} sk_i^{(j)}$, and the aggregated secret key is $sk = \sum_{j\in\mathbb{J}} sk^{(j)}$.

Unfortunately, the *delivery* phase alone currently involves *n broadcast* instances via Byzantine broadcast (BB) protocols [29]. Without a common coin [17], deterministic BB protocols generate $\Omega(t^2)$ messages [28]. Ensuring participation from at least $t+1$ participants is vital for sk's secrecy, causing the DKG's cubic communication for any $t = \Theta(n)$ [3].

Interactive consistency. Interactive consistency (IC) [10, 52] is a fundamental Byzantine fault-tolerant primitive, originally introduced in Pease et al.'s seminal work [52]. It allows a group of nodes, each with private inputs, to have a consistent view of a vector of everyone's inputs, which is essential for any multiparty computation expected to perform over everyone's input.

IC has an immediate reduction to BB: letting all nodes parallelly broadcast their inputs will allow everyone to have a consistent vector of inputs. Therefore, IC is also referred to as *parallel broadcast* in the literature [3, 59]. To the best of our knowledge, despite the recent advancements regarding randomized IC [59], n parallel deterministic BB remains the only solution for deterministic IC, which necessitates cubic communication cost.

The use of common coin. If assuming a common coin, the cubic barrier of both DKG and IC may be circumvented, for example, by sampling a committee to reduce the number of VSS instances (and thus the number of BB instances) or, more generally, by using randomized sub-quadratic BB protocols [1, 44] that are essentially committee-based. However, besides the fact that certain deterministic protocols already forbid it or potential adaptive security concerns, this workaround raises another crucial question: How can a common coin be established within the group (by those participants collectively)? Indeed, DKG is one of the main approaches to establishing a common coin, which leads to circularity. Meanwhile, for the non-DKG-based common coin protocols [11, 12, 25, 57] that do not rely on external setups or initial coins, a single execution of them already incurs $O(n^3)$ communication cost, which cannot help us to circumvent the cubic barrier from the ground.

In this paper, we are addressing the following long-standing problems:

Can we develop a DKG and/or IC protocol with sub-cubic communication complexity?

1.1 Our Results

In the paper, we answer the above question affirmatively by presenting a set of deterministic techniques, which we call *Dragon*, to efficiently replace the n parallel deterministic broadcast, thus finally getting rid of the cubic communication in both DKG and IC. We summarize our main results below.

New methodology: Dragon. In this paper, we present *Dragon*, short for *Decentralization at the cost of Representation after Arbitrary GrOupiNg*, which is a set of techniques for emulating a committee of representatives in a fully decentralized manner, without much overhead. Particularly, Dragon techniques do not need to assume common coins and do not pose any "single-point of failures". At a high level, it starts with an arbitrary deterministic grouping, which partitions the whole population into disjoint groups. Then, it comes with new techniques, including consortium-sender Byzantine broadcast (CSBB) and consortium-dealer secret sharing (CDSS) (we will elaborate more soon), which enable a group to broadcast a message or deal a secret at the cost of a single sender. We remark that using a small number of nodes to simulate a large number of nodes is a well-established technique for studying the possibility or impossibility of distributed computation tasks [15, 42]; this

[3]While we focus on synchronous DKG, asynchronous DKGs similarly let each participant invoke an asynchronous variant of broadcast protocol whose communication cost is $\Omega(n^2)$ [16], and consequently, all those asynchronous DKGs [2, 27, 34] also require $O(n^3\lambda)$ communication cost.

work explores the opposite direction of simulating a small number of nodes using a large number of nodes, for reducing the communication complexity.

Table 1. Comparison with the state-of-the-art synchronous DKG protocols.

Schemes	Resi.	Field?	PV?	Comm. Cost (total) w.oracles	Comm. Cost (total) opt.imp.	Comp. Cost (per node)	Round
Pedersen [53]	1/2	✓	✗	$O(n \cdot \mathsf{BB}_n(n\lambda))$	$O(n^3\lambda)$	$O(n^2)$	$O(\Delta_{\mathsf{BB}(n)})$
KZG[43](G.)	1/2	✓	✗	$O(n \cdot \mathsf{BB}_n(\lambda))$	$O(n^3\lambda)$	$O(n \log n)$	$O(\Delta_{\mathsf{BB}(n)})$
KZG[43](B.)				$O(n \cdot \mathsf{BB}_n(n\lambda))$	$O(n^3\lambda)$	$O(n^2)$	
FS [33]	1/2	✓	✓	$O(n \cdot \mathsf{BB}_n(n\lambda))$	$O(n^3\lambda)$	$O(n^2)$	$O(\Delta_{\mathsf{BB}(n)})$
GJM+ [41]	$\frac{\log n}{n}$	✗	✓	$O(n\mathsf{BB}_n(\lambda)+$ $\log n \cdot \mathsf{BB}_n(n\lambda))$	$O(n^3\lambda)$	$O(n \log^2 n)$	$O(\Delta_{\mathsf{BB}(n)})$
SBKN [56]	1/2	✓	✓	$O(n^3\lambda)$		$O(n^2)$	$O(n)$
Dragon-DKG (G)	1/2	✗	✓	$O(n \cdot \mathsf{BB}_{\sqrt{n}}(n\lambda)$ $+\sqrt{n} \cdot \mathsf{BB}_n(n\lambda))$	$O(n^{2.5}\lambda)$	$O(n^{1.5})$	$O(\Delta_{\mathsf{BB}(n)})$
Dragon-DKG (F)	1/2	✓	✓	$O(n \cdot \mathsf{BB}_{\sqrt{n}}(n\lambda)$ $+\sqrt{n} \cdot \mathsf{BB}_n(n\lambda\kappa))$	$O(n^{2.5}\lambda\kappa)$	$O(n^{1.5}\kappa)$	$O(\Delta_{\mathsf{BB}(n)})$

λ: computational security parameter; κ: statistical security parameter
`Resi.`: the maximal fraction of byzantine nodes the protocol can tolerate.
`Field?` asks if the secret key is in a finite field $\mathbb{Z}_p$ for some prime p.
`PV?` asks if the transcripts of the protocol are publicly verifiable.
`Round`: $\Delta_{\mathsf{BB}(n)}$ is the number of rounds of a Byzantine broadcast among n parties.
`Comp.Cost` measures the number of group exponent operations performed by each node. We assume the optimization from [19] has been applied whenever applicable.
`Comm.Cost` measures the number of bits sent by all honest nodes. In `w.oracles`, the cost is analyzed with oracle calls to BB/BA.
In `opt.imp.`, we calculate the cost with optimal BB and BA protocols. Some caveat exists, nevertheless, our claim of *sub-cubic* communication still holds, and we discussed in detail in Sect.3.1.
In KZG (G.), we analyze the cost of the optimized KZG protocol [63] when there is no complaint; In KZG (B.), we analyze the cost when there are $O(n^2)$ complaints.

Sub-cubic DKGs. We present two DKG protocols. We compare them with the state-of-the-art efficient DKGs in Table.1 and review more related works in Sect.2.

Our first DKG produces a secret key in a pairing-friendly Elliptic Curve group (given currently available instantiations). It has the communication complexity of

$$O(n^{2.5} \cdot |\mathsf{w}| + n \cdot \mathsf{BB}_{\sqrt{n}}(n\lambda) + \sqrt{n} \cdot \mathsf{BA}_n(n\lambda)),$$

where n is the number of participants, $|\mathsf{w}|$ is the size of membership witness of a cryptographic accumulator [50], and $\mathsf{BB}_k(\ell)$ (or $\mathsf{BA}_k(\ell)$) represents the communication cost of Byzantine broadcast (or Byzantine agreement (BA)) with k nodes on input of ℓ bits. Using optimal BB/BA [47, 48][4] and an accumulator with $|\mathsf{w}| = O(\lambda)$[50] (which assumes a CRS), the communication cost is $O(n^{2.5}\lambda)$[5]. In contrast, previous constructions incurs communication cost of $O(n \cdot \mathsf{BB}_n(n\lambda))$ or $O(n^3\lambda)$. It satisfies the *key-expressability* due to [41] and can thus be used to instantiate the key generation of the verifiable unpredictable function in [41]. Moreover, this DKG can tolerate up to $n/2$ Byzantine

[4]We discuss the instantiations of BA/BB in detail in Sect.3.1 and here we assume we have optimal BA/BB elsewhere, *i.e.*, $\mathsf{BA}_k(\ell) = \mathsf{BB}_k(\ell) = k\ell + k^2\lambda$.
[5]We note that we just use $n^{2.5}$ complexity here to demonstrate the power of our DRAGON method without much optimization, we can certainly further reduce it by the natural method of recursively applying DRAGON.

Table 2. Comparison with common coin rotocols without a strong setup

Techniques	Schemes	Resi.	Comm. Cost total	Comp. Cost (per node)
Time-lock Puzzl.	TCLM [57]	$\frac{n-1}{n}$	$O(n \cdot \mathsf{BB}_n(\lambda))$	Time-lock Puzzl.
PVSS	Scrape[19]	1/2	$O(n \cdot \mathsf{BB}_n(n\lambda))$	$O(n^2)$
Leader-PVSS	OptRand[11]	1/2	$O(n^3\lambda)$	$O(n^2)$
DKG	Ours	1/2	$O(n \cdot \mathsf{BB}_{\sqrt{n}}(n\lambda)$ $+\sqrt{n} \cdot \mathsf{BA}_n(n\lambda))$	$O(n^{1.5})$

nodes (which is optimal resilience) and has *publicly verifiable* transcripts (*i.e.*, not need a complaint phase). Moreover, it only requires each participant to perform $O(n^{1.5})$ group operations, which is superior to all prior work that needs $O(n^2)$ group operations (when considering optimal resilience).

Our second DKG produces a secret key in the finite field $\mathbb{Z}_p$ for some secure prime p, at the cost of increasing the communication/computation complexity by a factor of the statistical security parameter κ, while achieving the same security guarantee. This protocol can be a drop-in replacement for ElGamal encryption, BLS signature [8], Schnorr signature [55], and more [39].[6]

Both DKG constructions rely on publicly verifiable secret sharing (PVSS). Existing works formalize PVSS's secrecy with a game-based definition, which cannot be used to argue the security of DKG in a black-box way. As an independent interest, we present simulation-based definitions for PVSS (in Sect. 5), which allow us to analyze the security of our DKG in a modular manner. [7]

A simple corollary: a sub-cubic common coin protocol. Our DKG protocol gives rise to a sub-cubic common coin protocol. Concretely, to produce a common coin, a group of n participants initiates our DKG protocol. Upon the DKG's completion, participants exchange their secret shares amongst themselves. Subsequently, they can reconstruct the secret key and derive the common coin by hashing this key. Given that the adversary lacks prior knowledge of the secret key, the coin remains unpredictable and unbiased in the random oracle model. Table.2 [8] contrasts this construction with other common coin protocols that don't rely on a strong setup.

Sub-cubic IC. We present the first deterministic IC protocol with a message complexity of $O(n^{2.5})$ and communication complexity of $O(n^{2.5}\ell + n^{2.5} \cdot |\mathrm{w}| + n \cdot \mathsf{BB}_{\sqrt{n}}(n\lambda) + \sqrt{n} \cdot \mathsf{BA}_n(n\lambda))$. This can be further reduced to $O(n^{2.5}\ell + n^{2.5}\lambda)$ using optimal BB/BA and an $O(\lambda)$-sized accumulator. Our IC achieves strong adaptive security as per [1]. In contrast, existing deterministic IC protocols relying on n parallel deterministic BB require $O(n^3)$ message complexity and $O(n^2\ell + n^3\lambda)$ communication complexity. From a communication complexity perspective, our new IC has a clear advantage, especially when the input size is small, e.g., $\ell = O(\lambda)$. For information on randomized IC protocols, we refer to the comprehensive overview by Abraham et al. [4, Section 7.1].

1.2 Our Techniques

In the following, we use IC as an example to illustrate the core ideas behind Dragon and then show how to generalize it to obtain sub-cubic DKGs.

[6][40] showed that key-expressible DKG suffices for re-keyable primitives including ElGamal and BLS. While Schnorr signature is not re-keyable, its key generation algorithm can also be securely instantiated with a key-expressible DKG, as recently discussed in [39].

[7]We establish the static security of our DKGs. Our group-element DKG can be adaptive if using adaptively secure aggregatable PVSS [9], while our field-element DKG has no adaptively secure instantiations due to no suitable PVSS constructions.

[8]Notably, we omit various randomness beacon schemes, such as those based on PoW/VDF [60], as they pre-suppose an initial coin, making them unsuitable for single-shot common coin generation.

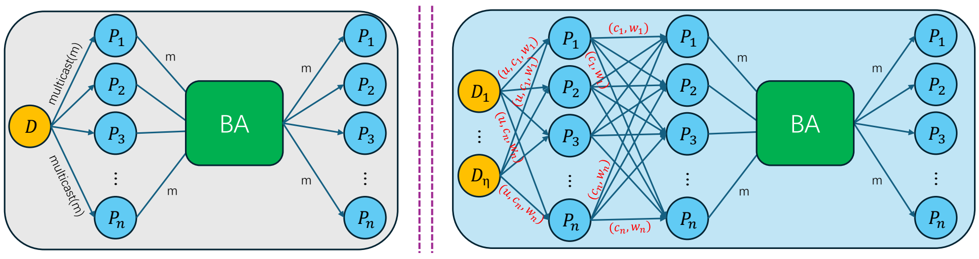

Fig. 1. The execution overflow of a conventional Byzantine broadcast (left side) and CSBB (right side).

Warm-up: efficient IC based on representatives. Our starting point is IC can be efficiently realized with the help of a committee of *representatives*, as long as *at least one* of them remain honest. Particularly, each node can first send its input with a signature to *all* representatives, such that every representative has the inputs of all honest nodes. Then, each representative *broadcasts* what it has received via BB to all nodes. Note that all nodes have the same view on the broadcasted messages; thus, each node can locally deduplicate the received broadcasted messages and finally agree on the same vector of input values. Since an honest representative will broadcast all received messages, the input of any honest node must be included in the broadcasted messages. Meanwhile, as every node signed its input, the input of an honest node will not be dropped during deduplication. Note that such a group of representatives can be obtained by using a common coin to randomly sample $O(\kappa)$ nodes from the whole population. κ is the statistical security parameter, and the probability of no honest node being selected is negligible in κ.

Dragon technique: CSBB. As sketched above, our methodology is to emulate a committee of representatives in a fully decentralized manner. We start with an arbitrary deterministic grouping, which ensures at least one group has an honest majority. We develop techniques to make each group "behave" like a single representative, in terms of both security and efficiency.

For IC, the most critical part is to let each group efficiently broadcast the input vector to the whole population. Trivial methods will fail: Letting everyone broadcast ruins our efforts to reduce communication, while if the task is assigned to only a small number of nodes, they could all be malicious. To tackle this issue, we formulate a new primitive called consortium-sender byzantine broadcast (CSBB) for a consortium of senders to broadcast to a larger population at the cost of a one-sender broadcast. I.e., broadcasting L-bit to the population of n nodes incurs the communication complexity of $O(nL + n^2\lambda)$.

Now, we elaborate on the ideas of CSBB. As illustrated in Fig. 1 (left side), a conventional BB protocol typically comprises a delivery phase, where the sender multicasts the message to receivers, and an agreement phase, where all receivers agree on the same message via a Byzantine agreement (BA) protocol. Inspired by (but distinct from) Byzantine agreement (BA) extension protocols [46], we employ error-correction codes [13] to deduplicate communications and reduce complexity. As depicted in Fig. 1 (right side), initially, all senders $D_1, \ldots, D_\eta$ locally encode their input messages using a deterministic encoding algorithm into n small fragments $(c_1, \ldots, c_n)$. Subsequently, the senders transmit each c_i to the corresponding P_i. Although P_i receives numerous fragments from different senders, most fragments should be identical under the assumption that honest senders have the same inputs. Consequently, P_i can deduplicate them and only multicast the most frequently appearing fragment to all other receivers. Finally, each receiver can collect enough fragments to reconstruct the message, simulating the scenario where an honest sender multicasts the message to all receivers. To assist a receiver in identifying the correct fragments, we employ a cryptographic accumulator [50], ensuring that fragments c_i and c_j generated from the same message m share the same accumulator value u and possess valid proofs w_i and w_j, respectively. For further details, please refer to Section 4.1.

Efficient IC from CSBB. After deterministic grouping, we let each group emulate a single representative in the warm-up committee-based IC, which yields an efficient deterministic IC. First, to emulate a representative, a group needs to form an input vector containing all honest nodes' input values. For this task, we let each node broadcast the received inputs *within* the group it belongs to, and then all nodes in the subgroup can agree on the same vector. Then, the group broadcasts the vector to the whole population via CSBB. Please see Sect.4.2 for more details.

Dragon technique for DKG: CDSS. Similar to IC, DKG can be greatly simplified with the help of a committee containing at least one honest representative. Particularly, as recently explored by Feng et al. in [31], only having these representatives as VSS dealers is sufficient for both the secrecy and robustness of the resulting DKG protocol. In this work, we develop *Dragon* techniques for emulating the representatives in a deterministic manner, thus removing the reliance on external common coins in [31] while preserving the efficiency benefits. Our main technical tool is to make each group "behave" like a single node to contribute one secret to the whole population, in terms of both security and efficiency. Specifically, for security, the secret contributed by a good group should remain *secret* to the adversary. We formulate a new secret-sharing primitive CDSS to capture this functionality (in Sect.6.1). We then show a DKG can be built by concurrently executing multiple CDSS instances where each group acts as a dealer consortium. Please see Sect.6.4 for details. We present two CDSS protocols for group-element secrets and field-element secrets, respectively.

CDSS with group-element secrets. Constructing an efficient CDSS requires conquering the following challenges: first, similar to conventional DKG, when nodes (now in each group) jointly contribute one secret $sk^{(i)}$, they need to distribute each of their shares to the whole population for future reconstruction; but at the same time, they cannot directly reveal those shares — they together can be used to reconstruct $sk^{(i)}$, which might be the only "good" secret (thus adversary can recover the final sk without enough shares).

We leverage the *aggregatable* PVSS [19, 41] to address this challenge.[9] PVSS produces a sequence of encrypted shares under receivers' public keys with proof of well-formedness of ciphertexts; aggregatable PVSS allows the compression of many transcripts (including the ciphertexts and proofs) into one. We let each node in the group generate a PVSS transcript for the whole population but only broadcast it *within the group*. Then, everyone in the group aggregates the received PVSS transcripts into the *same* (because of the public verifiability) one transcript ($sk^{(i)}$ is not leaked at all), which is to be sent to the whole population. Please see Sect.6.2 for more details.

CDSS with field-element secrets. The challenge towards field-element DKG arises from the current incompatibility of aggregatable PVSS with field-element secrets. To address this, we adopt a more general approach, constructing a CDSS from a conventional (non-aggregatable) PVSS scheme. In our CDSS with group-element secrets, aggregatable PVSS is employed to reduce the message size broadcasted by the dealer group while ensuring the secrecy of the corresponding secret key against adversaries. We observe that both goals can be achieved using a common coin *within* the dealer group. Note that we only expect nodes within the same group, instead of the whole population, to agree on this common coin. Such an intra-group coin can be efficiently realized by running a DKG *within* the group without incurring $O(n^3)$ complexity. After the dealers broadcast their PVSS transcripts within the group, a common coin is employed to determine a small number (denoted by κ, linear

[9]We remark aggregatable PVSS has been leveraged recently to reduce the cost of DKG by Gurkan et al. [41] from $n\mathsf{BB}_n(n\lambda)$ to $n\mathsf{BB}_n(\lambda) + \log n \cdot \mathsf{BB}_n(n\lambda)$ (but it is still $O(n^3\lambda)$, and with the price of lowering resilience to $O(\log n/n)$). We take a different approach to using aggregatable PVSS together with our dragon approach, finally breaking the cubic barrier.

to the statistical security parameter) of valid transcripts. This ensures that at least one of them is from an honest dealer with overwhelming probability. The dealer group then broadcasts all selected transcripts to all receivers via CSBB. Finally, the receiver decrypts all transcripts and computes the sum of the decrypted values as the received share. As at least one transcript is from an honest dealer, the secret key remains unknown to the adversary. However, since the dealer group now needs to broadcast κ PVSS transcripts, the communication cost is higher than that of the aggregatable PVSS-based approach by a factor of κ. Please refer to Sect. 6.3 for detailed information.

2 RELATED WORKS

Distributed Key Generation. DKG has emerged as a significant research domain over the decades. Pedersen [53] laid the groundwork in this area by presenting the first efficient protocol for Dlog-based cryptosystems, which is built upon Feldman's Verifiable Secret Sharing (VSS) [30]. In Pedersen's approach, all users collaboratively execute n instances of Feldman's VSS, with each user acting as the dealer for one instance.

Within Feldman's VSS framework, the dealer must broadcast a commitment to a polynomial while privately dispatching the shares to all other users. Given that the commitment size is $O(n\lambda)$, the communication overhead stands at $O(n\mathsf{BB}_n(n\lambda))$, where $\mathsf{BB}_n(\ell)$ represents the communication cost for executing a Byzantine broadcast protocol among n nodes with an ℓ-bit input. Furthermore, Pedersen's DKG necessitates a complaint phase where users broadcast their complaints against dishonest dealers. Considering that a user might broadcast multiple complaints simultaneously, the communication overhead for this phase also matches $O(n\mathsf{BB}_n(n\lambda))$. It's crucial to note, however, that during this complaint phase, each user might verify up to $O(n^2)$ shares. For Feldman's VSS, the computational cost associated with verifying a single share amounts to $O(n)$ group operations. Therefore, the per-node computational overhead before the complaint phase is $O(n^2)$, which can escalate to $O(n^3)$ in the complaint phase.

Most DKG constructions adhere to the joint-VSS paradigm. Essentially, every novel VSS protocol can be transformed into a new DKG protocol. Moreover, since VSS can be established on polynomial commitments, every polynomial commitment scheme can also culminate in a VSS and, ultimately, a DKG. Kate et al. [43] proposed the first polynomial commitment (denoted as KZG) with an $O(\lambda)$ commitment size. Consequently, prior to the complaint phase, the communication cost can be tapered down to $O(n\mathsf{BB}_n(\lambda))$. This, however, is still asymptotically $O(n^3\lambda)$, even when paired with an optimal broadcast protocol. An added advantage of the KZG polynomial commitment is its efficiency in share verification; verifying a single share incurs a mere $O(1)$ cost. This implies that the per-node computational cost for verification before the complaint phase is a mere $O(n)$ in group operations, though it can inflate to $O(n^2)$ during the complaint phase. While the computational overhead for generating the polynomial commitment was believed to be $O(n^2)$ [58], a novel study by Zhang et al. [63] demonstrated that the computational overhead for generating a KZG commitment can be optimized to $O(n \log n)$. Though KZG mandates a CRS setup, other endeavors [61, 63] focusing on efficient polynomial commitments without a trusted setup have been explored, but they fall short of KZG's efficiency.

Fouque and Stern [33] circumvented the need for a complaint phase by integrating publicly verifiable secret sharing (PVSS). Given that a PVSS transcript encompasses $O(n)$ ciphertexts, the communication cost invariably aligns with $O(n\mathsf{BB}_n(n\lambda))$, should all users opt to broadcast the transcript. Traditionally, verifying a PVSS transcript demanded an $O(n^2)$ overhead, implying that the per-node computational cost in DKG could reach $O(n^3)$. This hurdle was overcome by Cascudo and David in Scrape [19], which introduced a PVSS scheme that limits verification time to $O(n)$. Notably, Scrape's methodology is versatile and can be applied to enhance several preceding techniques, including Pedersen's, ensuring that the computational overhead during the complaint phase remains at $O(n^2)$ rather than surging to $O(n^3)$. A dedicated line of research, evident in works like [37], has aimed to refine the concrete performance of PVSS.

Gurkan et al. [41] harnessed an aggregatable PVSS in tandem with gossip protocols to devise a publicly verifiable DKG. Their communication cost is $n\mathsf{BB}_n(\lambda) + \log n \cdot \mathsf{BB}_n(n\lambda)$ (still $O(n^3\lambda)$) instead of $n\mathsf{BB}_n(n\lambda)$, and their per-node communication cost is $O(n \log^2 n)$. However, their scheme can only tolerate $O(\log n)$ Byzantine nodes. Shrestha et al. [56] charted a different path, presenting a DKG without resorting to BB protocols. Instead, they employed an MVBA [46] protocol to facilitate agreement, which culminates in an $O(n^3\lambda)$ communication overhead. They posited that achieving optimal resilience with BB without a private setup should require $O(n^3)$ communication cost. Yet, in light of recent advancements in transparent threshold signatures [5], this hypothesis might need reevaluation. We delve deeper into the intricacies of BB/BA in Sect.3.1, while elsewhere, we assume the existence of an optimal BA/BB.

Beyond these efforts directed at augmenting the efficiency of DKG, there have been other studies addressing this challenge using distinct criteria. Gennaro et al. [36] discerned that the secret key distribution in Pedersen's DKG could be influenced by adversarial entities. They rectified this shortcoming, achieving full secrecy but at a higher computational cost. Gurkan et al.[41] introduced a weaker version of secrecy termed “key-expressability", which postulates that adversaries might influence key distribution but within confined parameters. They argued that a key-expressable DKG suffices for a plethora of applications, with numerous DKG frameworks, including those of Pedersen [53], Fouque-Stern [33], and our own, aligning with this definition. Canetti et al. [18] contributed a DKG protocol with adaptive security, a contrast to our model and several others that ensure security against only static adversaries. Recent contributions by Bacho and Loss [8] introduced an oracle-aided adaptive definition and verified that multiple protocols align with this definition in the algebraic group model.

Lastly, a few recent endeavors [2, 27, 34] have shifted the focus towards DKG in asynchronous networks. These constructions adopt the joint-VSS framework and rely on an asynchronous broadcast protocol termed “reliable broadcast" [16] to ensure verifiability, consequently still facing the cubic computational barrier. Notably, Das et al. [27] presented the pioneering asynchronous DKG with an $O(n^3\lambda)$ communication overhead for field-element secrets, while Abraham et al.[2] delivered an adaptively secure asynchronous DKG with same complexity.

Distributed Common Coin. A common coin protocol allows a group of participants/nodes to produce unbiased and unpredictable common randomness, which is paramount to many applications, such as lottery [14], committee sampling in distributed protocols [62], and asynchronous consensus [32]. In history, the line of common coin study has been closely related to, yet in parallel with, DKG.

A major approach to the common coin is called *commit-then-reveal*, where each participant P_i first commits a randomness r_i to all others and then reveals r_i such that the coin $r = \sum r_i$. To prevent an attacker from withholding a commitment (after seeing other r_i) to bias the coin, we will need a commitment scheme supporting forced opening, such as publicly verifiable secret sharing (PVSS) [19, 33] and time-locked commitment [57]. This approach similarly requires each participant to *broadcast* its commitment; those $O(n)$ broadcast instances immediately cause $\Omega(n^3\lambda)$ communication cost again when implementing with deterministic BB protocols (now, there is no coin to use).

A recent line of research [11, 12, 25] discards the expensive broadcast procedures and uses a leader node to coordinate the communication; with an honest leader, the group can produce a common coin at the communication cost of $O(n^2\lambda)$. However, as leaders are switched in the Round-Robin manner, only after the $t + 1$ leader election can we guarantee an honest leader. Therefore, for a single-shot coin generation, the leader-based approach still incurs $O(n^3\lambda)$ communication. Moreover, there are batched coin protocols, including [20, 21], which may generate up to $O(n^2)$ common coins in a single execution. Achieving low amortized complexity is an interesting orthogonal approach, and we leave a batched protocol based on our subcubic single-shot one as an interesting future work. However, a single execution of these batched coin protocols will incur cubic communication already, which cannot help us circumvent the cubic complexity in DKG and IC.

Another natural solution for the common coin is letting the group execute a DKG protocol and recover (the hash of) the secret key as the coin. Indeed, as suggested in Cachin et al.'s pioneering work [17] and adopted by many real-world projects like Drand[10], we may use (the hash of) a unique threshold signature as the common coin, after a DKG or trusted key generation for setting up the signing keys. Unfortunately, these approaches based on DKG create a *circular* problem.

Comparison with Concurrent Work. In a very recent concurrent work [7] [11], Bacho et al. also introduced a DKG with sub-cubic communication complexity. Here, we provide a comparison between the two works.

Regarding techniques, the one introduced in [7], and ours differ significantly. Their DKG employs dedicated consensus to agree on aggregated PVSS transcripts, while ours leverages arbitrary grouping together with consortium dealer secret sharing (and broadcast), which can be built upon any BB/BA in a black box manner. These underlying techniques may find different applications beyond DKGs. As we have shown in this paper, our techniques naturally give rise to a deterministic IC with sub-cubic communication complexity.

Also, these different techniques may lead to different efficiency, security, and functionality trade-offs, as briefly elaborated below.

In terms of communication complexity, their DKG (with a communication complexity of $O(n^2 \log n\lambda))$ asymptotically outperforms our current constructions. We didn't do further optimization when getting communication down to sub-cubic; in principle, our techniques may be applied recursively and further bring down the communication to be arbitrarily close to n^2.

Regarding functionality, we provide sub-cubic DKGs for both group-element secrets and field-element secrets, while [7] only demonstrates a DKG with group-element secrets. It is worth noting that it is possible to use the DKG in [7] as a common coin to construct a communication-efficient DKG with field-element secrets, for instance, by sampling a committee of dealers [31]. However, the resulting scheme cannot be strongly adaptively secure [1], even assuming an adaptively secure PVSS with field-element secrets. In contrast, our field-element DKG can be strongly adaptively secure, assuming an adaptively secure PVSS and memory erasures, though how to construct an efficient and adaptively secure PVSS with field-element secrets remains open.

In terms of security, the two results are generally incomparable. While their primary application is a randomness beacon, the DKG in [7] is proven to be *unpredictable* against *adaptive* adversaries in the algebraic group model. In contrast, our focus is on applications to threshold cryptography, and we prove that our DKG satisfies *key-expressibility* against *static* adversaries in the standard model (while our instantiations may assume a random oracle). However, we believe there is no significant gap between the security guarantees provided by the two works. Our DKG schemes can be adaptively secure if the underlying components, particularly the PVSS scheme, are adaptively secure. By utilizing the adaptive PVSS scheme introduced in [9], our group-element DKG can achieve the same security guarantee as the DKG in [7]. Conversely, if focusing on static security, their DKG may achieve the stronger *key-expressibility*, although further analysis is required.

[10]https://drand.love/

[11]A previous version of our paper was submitted to a conference in 2023 before [7] was available online in December.

3 MODEL, PRELIMINARIES, AND PROTOCOL COMPOSITION

Notations. Throughout the paper: We use λ to represent the security parameter. The notation $[i, n]$ represents the set $\{i, i+1, \cdots, n\}$, where i and n are integers with $i < n$. We might abbreviate $[1, n]$ simply as $[n]$. For a set $\{x_1, x_2, \ldots, x_n\}$ and a sequence $(x_1, x_2, \ldots, x_n)$, we may abbreviate them as $\{x_i\}_{i\in[n]}$ and $(x_i)_{i\in[n]}$, respectively. If $\odot$ represents a group operation in $\mathbb{G}$, then $g_1 \odot g_2 \cdots \odot g_n$ is denoted as $\bigodot_{i\in[n]} g_i$ where each $g_i \in \mathbb{G}$. An execution of the protocol Π involving n participants P_i, each with inputs v_i, is represented by $\Pi\langle\{P_i(v_i)\}_{i\in[n]}\rangle$. We call an integer p a secure prime if it is sufficiently large such that the DLog problem in the corresponding group of order p is hard.

A function $f(n)$ is deemed negligible in n, denoted by $f(n) \leq \mathrm{negl}(n)$, if for every positive integer c, there exists an n_0 such that for all $n > n_0$, $f(n) < n^{-c}$. Conversely, a non-negligible function is denoted as $f(n) > \mathrm{negl}(n)$. For a set $\mathbb{X}$, the notation $x \leftarrow\!\!\$\ \mathbb{X}$ signifies sampling x uniformly from $\mathbb{X}$. Given a distribution X, $x \leftarrow X$ denotes sampling x from X. For a probabilistic algorithm A, $A(x_1, x_2, \cdots; r)$ represents the result of running A with inputs $x_1, x_2, \cdots$ and random coins r. We use $y \leftarrow A(x_1, x_2, \cdots)$ to represent choosing r randomly and obtaining $y = A(x_1, x_2, \cdots; r)$. If $\odot$ represents a group operation in $\mathbb{G}$, then $g_1 \odot g_2 \cdots \odot g_n$ is denoted as $\bigodot_{i\in[n]} g_i$ where each $g_i \in \mathbb{G}$. An execution of the protocol Π involving n participants P_i, each with inputs v_i, is represented by $\Pi\langle\{P_i(v_i)\}_{i\in[n]}\rangle$. Adversaries are assumed to be probabilistic polynomial time (PPT).

Communication and threat model. We assume the network is synchronous, and the protocol proceeds in rounds. The network is fully connected, meaning there is a communication channel between each pair of nodes. We assume the channel is authenticated, and we measure communication complexity by the number of bits sent by honest nodes.

We assume an initial phase that optionally generates a common reference string (CRS) and sets up PKI for every participant. We consider both *static* and *adaptive* adversaries. A static adversary needs to specify the set of corrupted nodes at the beginning of the system after seeing CRS, while an adaptive adversary can adaptively corrupt nodes at any time during the protocol execution. Once a node gets corrupted, the adversary gets access to its local states and controls its subsequent behaviors. Particularly, an adaptive adversary can perform the "after-fact-removal" attacks [47], *i.e.*, corrupt a node and *remove* all messages the node just sent before being delivered. For both cases, we assume the total number of corrupt nodes at the end of the execution is at most t.

3.1 Byzantine Consensus

Byzantine Agreement. In an (n, t, ℓ)-Byzantine Agreement (BA) protocol, there are n parties $P_1, \ldots, P_n$, each P_i having an ℓ-bit initial input v_i, denoted as $\mathsf{BA}\langle P_i(v_i)\rangle$. Against any adversary $\mathcal{A}$ that corrupts up to t parties, a secure BA ensures the following properties:

- **Validity.** If all honest parties share the same input v, they all output v.
- **Agreement.** All honest parties output the same message.
- **Termination.** All honest parties produce an output message.

Byzantine Broadcast. In an (n, t, ℓ)-Byzantine Broadcast (BB) protocol, there is a sender P_s with ℓ-bit input message $\mathtt{msg}$ and a set of receivers $\mathcal{P} = \{P_1, \ldots, P_n\}$, denoted as $\mathsf{BB}\langle P_s(\mathtt{msg}), \mathcal{P}\rangle$. A secure BB has the same agreement and termination guarantee as a BA does, but it concerns the following validity:

- **Validity.** If the sender is honest, all honest receivers output the sender's input message $\mathtt{msg}$.

Instantiations of BA and BB. We first examine the candidates of BA protocols. For ease of reference, we use $\mathsf{BA}_n(\ell)$ to denote the communication complexity of a BA protocol among n parties with ℓ-bit input. We summarize the status in Table.3 and elaborate on them in the following.

Table 3. Summary of BA Protocols

Candidates	$\mathsf{BA}_n(\ell)$	Resilience	Setup	Computation
B1 + E1	$O(n\ell + n^2\lambda)$	$t < (1/2 - \epsilon)n$	CRS	`light`
B1 + E2	$O(n\ell + n^2 \log n\lambda)$	$t < (1/2 - \epsilon)n$	Transparent	`light`
B2 + E2	$O(n\ell + n^2 \log n\lambda)$	$t < n/2$	Transparent	`moderate`
B3 + E1	$O(n\ell + n^2\lambda)$	$t < n/2$	CRS	`heavy`

In scenarios where $\ell = O(\lambda)$, Momose and Ren [47] have presented the state-of-the-art protocols. These include:

- A BA protocol (referred to as B1) with $\mathsf{BA}_n(\lambda) = O(n^2\lambda)$, capable of tolerating $t < (1/2-\epsilon)n$ corruptions, for any positive constant $\epsilon \in (0, 1/2)$. This protocol requires only conventional digital signatures.
- Another BA protocol with $\mathsf{BA}_n(\lambda) = O(n^2 \cdot \mathsf{thrSigSize})$, tolerating $t < n/2$ corruptions, where $\mathsf{thrSigSize}$ is the size of a threshold signature. Traditional threshold signatures require either trusted key generation or a DKG phase, which is unfavorable for our purpose. However, recent progress [5] provides a threshold signature of size $O(\lambda \log n)$ with a transparent setup (referred to as B2). Additionally, a constant-size zk-SNARK [38] can also yield a threshold signature of $O(\lambda)$ size (referred to as B3).

For cases where $\ell > k$, Nayak et al. [48] have provided extension protocols. These result in $\mathsf{BA}_n(\ell) = O(n\cdot\ell + \mathsf{BA}_n(\lambda) + n^2\lambda)$ if a CRS for a pairing-based accumulator is allowed (referred to as E1). Alternatively, if the CRS is not allowed, the complexity becomes $\mathsf{BA}_n(\ell) = O(n\cdot\ell + \mathsf{BA}_n(\lambda) + n^2 \log n\lambda)$ (referred to as E2).

It has been suggested that $\mathsf{BA}_n(\ell) = O(n\ell + n^2\lambda)$ could be optimal for $\ell > \lambda$ in the PKI setting [48], despite the proved lower bound being $O(n\ell + n^2)$ [28]. Combining B3 and E1 could yield a BA protocol with optimal communication complexity and optimal resilience $t < \frac{n}{2}$, although it necessitates a CRS setup and potentially intensive computation. Other combinations may alleviate concerns about setup or computation, offering sub-optimal communication complexity or resilience.

For $t < n/2$, a BB protocol can be considered where the sender first multicasts its input to all receivers, and then the receivers execute a BA protocol to finalize their output. Thus, $\mathsf{BB}_n(\ell) = O(\mathsf{BA}_n(\ell))$. Suitable instantiations follow our discussion about BA protocols.

Nevertheless, the study of optimal BA/BB protocols is a rapidly evolving field and largely unrelated to our primary focus. Hence, for simplicity, in the remainder of this paper, we assume a BA protocol with optimal communication complexity and resilience and do not account for potential setup requirements and computational overhead.

Interactive Consistency (IC) [10]. In an (n, t, ℓ)-Interactive Consistency (IC) protocol, there are n parties $P_1, \ldots, P_n$, each P_i having an ℓ-bit initial input v_i, denoted as $\mathsf{IC}\langle P_i(v_i)\rangle$. The protocol involves n nodes, where up to t nodes are corrupted. It aims to let each node output the same vector if all honest nodes have an input. Formally, against any adversary $\mathcal{A}$ that corrupts up to t parties, it satisfies the following properties:

- **Agreement.** If any two honest nodes output, then their output vectors must be the same.
- **Validity.** If an honest party outputs a vector S, then $|S| \geq n - t$ and S contains the input values from all honest nodes.
- **Termination.** All honest parties produce an output message.

3.2 Distributed Key Generation

Homomorphic Key Structure. We consider a generic homomorphic key structure. Let $\mathcal{SK}$ represent the group of secrets with the operation $\oplus$, and $\mathcal{PK}$ denote the group of public keys with the operation $\otimes$. The structure includes a PPT algorithm KeyGen and a relation Rela $\subset (\mathcal{PK}, \mathcal{SK})$ such that

$$\Pr[(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda) : (pk, sk) \in \mathsf{Rela}] = 1.$$

We say that $(\mathcal{PK}, \mathcal{SK})$ forms a homomorphic key structure if Rela is homomorphic. That is, for any $pk_1, pk_2 \in \mathcal{PK}$ and $sk_1, sk_2 \in \mathcal{SK}$ such that $(pk_i, sk_i) \in \mathsf{Rela}$ $(i = 1, 2)$ and any integers $\alpha, \beta \in \mathbb{N}$, it holds that

$$(\alpha pk_1 \otimes \beta pk_2, \alpha sk_1 \oplus \beta sk_2) \in \mathsf{Rela}.$$

The key generation algorithm KeyGen should satisfy specific properties to be useful in cryptography. However, these specificities are not central to our discussion on securely decentralizing the algorithm.

Instantiation. Throughout this paper, we focus on two key structures. The first is the standard key structure in DLog-based cryptography. Here, the key generation algorithm yields $sk \leftarrow\!\!\$\; \mathcal{SK} := \mathbb{Z}_p$ and $pk = g^{sk} \in \mathcal{PK} := \mathbb{G}$, where p is a secure prime and g is a generator of a cyclic group $\mathbb{G}$ of order p. The relationship $(pk, sk) \in \mathsf{Rela}$ holds iff $pk = g^{sk}$.

The second structure is a pairing-based one [41]. In this structure, the key generation algorithm creates $pk = (g^s, u^s) \in \mathbb{G}_1 \times \mathbb{G}_2$ and $sk = h^s \in \mathbb{G}_2$, where $s \leftarrow\!\!\$\; \mathbb{Z}_p$ for a particular secure prime p. $\mathbb{G}_1$ (with generator g) and $\mathbb{G}_2$ (with generators u and h) are cyclic groups of order p. A bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \longrightarrow \mathbb{G}_T$ exists. The relation $(pk = (pk', pk''), sk) \in \mathsf{Rela}$ holds iff $e(pk', h) = e(g, sk)$ and $e(pk', u) = e(g, pk'')$. Here, $pk_1 \oplus pk_2 = (pk_1' \cdot pk_2', pk_1'' \cdot pk_2'')$.

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

- $\mathsf{Init}(1^\lambda, n)$. It generates a CRS `crs` and establishes the PKI for $\mathcal{P}$.
- $\mathsf{Rec}((i, sk_i)_{i\in\mathbb{I}})$. Given a set of $t + 1$ key shares as input, it outputs the secret key sk for pk.

In this paper, we address Π_{DKG} meeting both *robustness* and *key-expressibility*, the latter being a weaker form of *secrecy*.

Robustness: Π_{DKG} is robust if, even when up to t parties are compromised, every honest P_i outputs the same $(pk, (pk_i)i \in [n])$, and its sk_i satisfies $(pk_i, sk_i) \in \mathsf{Rela}$. Additionally, for any two sets, $\mathbb{I}_1$ and $\mathbb{I}_2$, with $t + 1$ honest participants each, a unique secret key sk can be reconstructed from their secret shares, as described by the equation below:

$$\Pr\begin{bmatrix} \mathsf{Rec}(pk, (pk_i)_{i\in[n]}, \{sk_i\}_{i\in\mathbb{I}_1}) = \mathsf{Rec}(pk, (pk_i)_{i\in[n]}, \{sk_i\}_{i\in\mathbb{I}_2}) \\ \hfill \wedge (pk, \mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}_1})) \in \mathsf{Rela} \end{bmatrix} = 1.$$

Key expressibility: Π_{DKG} is *key expressable* if, for any PPT adversary $\mathcal{A}$ that compromises up to t nodes, a PPT simulator $\mathsf{Sim}^{\mathcal{A}}$ exists s.t. the following equation holds for any PPT distinguisher $\mathcal{A}'$:

$$\left| \Pr\begin{bmatrix} \Pi^{\mathcal{A}}_{\mathsf{DKG}}\langle\{P_i(1^\lambda)\}_{i\in[n]}\rangle \\ \longrightarrow (pk, \mathsf{view}_{\mathcal{A}}) : \\ \mathcal{A}'(pk, \mathsf{view}_{\mathcal{A}}) = 1 \end{bmatrix} - \Pr\begin{bmatrix} \mathsf{KeyGen}(1^\lambda) \longrightarrow (pk, sk), \\ \mathsf{Sim}^{\mathcal{A}}(pk) \longrightarrow (sk', pk', \\ \alpha \in \mathbb{Z}^+, \mathsf{sview}_{\mathcal{A}}) : \\ \mathcal{A}'(\alpha pk \otimes pk_1, \mathsf{sview}_{\mathcal{A}}) = 1 \\ \wedge\ (sk', pk') \in \mathsf{Rela} \end{bmatrix} \right| \leq \mathsf{negl}(\lambda),$$

where the notation $\Pi^{\mathcal{A}}_{\mathsf{DKG}}\langle\{P_i(1^\lambda)\}_{i\in[n]}\rangle \longrightarrow (pk, \mathsf{view}_{\mathcal{A}})$ represents an execution of Π_{DKG} involving the adversary $\mathcal{A}$, *including the initial phase*. Here, pk is the public key generated by the execution, and $\mathsf{view}_{\mathcal{A}}$ represents the adversary's view during the execution, including all public messages and its internal states; KeyGen is the default key generation algorithm for $(\mathcal{PK}, \mathcal{SK})$.

Remark 1. *The key-expressibility presented above is a more formal version of the definition in [41]. Key-expressibility is weaker than full secrecy [36] but still sufficient for instantiating the key generation algorithm for a variety of cryptosystems that are "re-keyable", including BLS signatures, ElGamal encryption, etc., as discussed in [41]. The definition can also be reformulated as an ideal functionality, known as BiasedKeyGen in [39]. Moreover, as demonstrated by Groth and Shoup [39], a DKG realizing BiasedKeyGen (or satisfying key-expressibility) can securely instantiate the key generation for (threshold) Schnorr signatures.*

Remark 2. *Some applications such as random coins may only require the DKG to be* ***unpredictable****, i.e., for any PPT adversary $\mathcal{A}$, it holds that.* $\Pr[\Pi^{\mathcal{A}}_{\mathsf{DKG}}\langle\{P_i(1^\lambda)\}_{i\in[n]}\rangle \to (pk, \mathsf{view}_{\mathcal{A}}), \mathcal{A}(\mathsf{view}_{\mathcal{A}}) \to sk : (pk, sk) \in \mathsf{Rela}] \le \mathsf{negl}(\lambda)$. *Unpredictability is clearly implied by key expressibility.*

3.3 Other Cryptography Primitives

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)$\mathsf{Vrfy}(ak, u, s_i, w_i)$ decides if s_i is an element accumulated into u, using the witness w_i.

An accumulator scheme is *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 $w_i \leftarrow \mathsf{Wit}(ak, u, s_i)$, 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,

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

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

Instantiation. We primarily consider the pairing-based accumulator [50] which requires a CRS and has $O(\lambda)$-sized witness. Merkle tree is also a candidate featured by its transparent setup, although the witness size is $O(\lambda \log n)$.

Erasure code scheme. A (k, n)-erasure code scheme [13] consists of two deterministic algorithms Encode and Decode. The Encode algorithm maps any vector $\mathbf{m} = (m_1, \cdots, m_k)$ of k data fragments into an vector $\mathbf{c} = (\mathsf{c}_1, \ldots, \mathsf{c}_n)$ of n coded fragments, such that any k elements in the code vector $\mathbf{c}$ is enough to reconstruct $\mathbf{m}$ due to the Decode algorithm. I.e., for any $\mathbf{m} \in \mathcal{B}^k$ and any $\mathbb{I} \subset [n]$ that $|\mathbb{I}| = k$, we have

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

Instantiation. Throughout the paper, we consider a $(t+1, n)$-erasure code where $2t+1 = n$. Additionally, it's important to note that this erasure code scheme will implicitly select an appropriate $\mathcal{B}$ based on the actual length of each element in $\mathbf{v}$. This ensures that the encoding results in only a constant size increase.

NIZK. A non-interactive zero-knowledge (NIZK) proof system Π, for an NP language L, enables the prover, who holds a witness of an instance $x \in L$, to convince the verifier that $x \in L$ via a single proof. Typically, it can be described by the following a triple of probabilistic polynomial-time (PPT) algorithms:

- $\sigma \leftarrow \mathsf{Setup}(1^\lambda)$. The setup algorithm outputs a CRS σ.
- $\pi \leftarrow \mathsf{Prove}(\sigma, x, w)$. The prover algorithm takes as inputs the CRS σ, an instance $x \in L$ with its witness $w \in R_L(x)$, and outputs a string π called a proof.
- $b \leftarrow \mathsf{Verify}(\sigma, x, \pi)$. The verifier algorithm takes as inputs σ, an instance x and a proof π, and outputs either 1 accepting it or 0 rejecting it.

We consider a NIZK satisfying *completeness*, *zero-knowledge*, and *simulation soudness*.

(1) Completeness: For all security parameters $\lambda \in \mathbb{N}$ and for all $x \in L_\lambda$ and $w \in R_L(x)$,

$$\Pr[\sigma \leftarrow \mathsf{Setup}(1^\lambda); \pi \leftarrow \mathsf{Prove}(\sigma, x, w) : \mathsf{Verify}(\sigma, x, \pi) = 1] = 1.$$

(2) Zero knowledge: There is a PPT simulator $(\mathsf{SimSetup}, \mathsf{SimProve})$, *s.t.* for every PPT adversary $\mathcal{A}$, we have

$$\Pr[\sigma \leftarrow \mathsf{Setup}(1^\lambda) : 1 \leftarrow \mathcal{A}^{O_1(\sigma,\cdot,\cdot)}(\sigma)] - \Pr[(\sigma, \tau \leftarrow \mathsf{SimSetup}(1^\lambda) : 1 \leftarrow \mathcal{A}^{O_2(\sigma,\tau,\cdot,\cdot)}(\sigma)] \leq \mathsf{negl}(\lambda).$$

Both the oracles O_1 and O_2 take as input a pair $(x, w) \in R_L(x)$. While O_1 returns $\pi \leftarrow \mathsf{Prove}(\sigma, x, w)$, O_2 returns $\pi \leftarrow \mathsf{SimProve}(\sigma, \tau, x)$.

(3) Simulation soundness: For any PPT adversary $\mathcal{A}$, it holds that

$$\Pr\left[\begin{array}{r}(\sigma, \tau) \leftarrow \mathsf{SimSetup}(1^\lambda); (x^*, \pi^*) \leftarrow \mathcal{A}^{O_2(\sigma,\tau,\cdot)}(\sigma) : \\ \mathsf{Verify}(\sigma, x^*, \pi^*) = 1 \wedge x^* \notin L\end{array}\right] \leq \mathsf{negl}(\lambda).$$

3.4 Unique Identifier Model

Our DKG constructions are built upon multiple sub-protocols, which could run concurrently. To ensure the security of our DKGs, each sub-protocol must remain secure during simultaneous operations. Lindell *et al.* [45] highlighted that many BA protocols retain their security in concurrent settings if each is given a *unique identifier*. We define this concept as follows.

Definition 3.1 (Unique identifier model.). Protocol Π uses a signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ to sign/verify its messages. In the unique identifier model, every instance of Π gets a distinct identifier id, leading to a modified protocol Π_{id}. Π_{id} is like Π, but it utilizes $\Sigma_{id} = (\mathsf{Gen}, \mathsf{Sign}_{id}, \mathsf{Vrfy}_{id})$, where $\mathsf{Sign}_{id}(sk, m) = \mathsf{Sign}(sk, id||m)$ and $\mathsf{Vrfy}_{id}(vk, m, \sigma) = \mathsf{Verify}(vk, id||m, \sigma)$. Different instances must have prefix-free id strings, ensuring one id isn't a prefix of another.

Simply put, protocols can maintain concurrent security in this model by disregarding messages with different identifiers, ensuring security akin to isolated settings. Most consensus protocols should be concurrently secure in this model.

4 DRAGAON-IC

Deterministic grouping. Conventional sharding relies on common randomness to ensure all shards have adequate, honest participants. This creates a circular issue for DKG/Coin, which is meant to establish such randomness. Instead, we propose a deterministic grouping, which divides the population into subgroups using an arbitrary predefined rule. While this method may not offer a strong guarantee, it still ensures that at least one group maintains the honesty ratio.

Lemma 4.1 (Any-good Partition). *For a population $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ and a partition $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_m\}$ over $[n]$. If there are t corrupted nodes, denoted as $\{P_i\}_{i \in C}$ where $C \subset [n]$ and $|C| = t$, then there exists a subset $\mathcal{S}_j$ such that the proportion of corrupted nodes in $\{P_i\}_{i \in \mathcal{S}_j}$, given by $\frac{|C \cap \mathcal{S}_j|}{|\mathcal{S}_j|}$ is at most $\frac{t}{n}$.*

Proof. Suppose that no subset $\mathcal{S}_j$ satisfies the condition, *i.e.*, $\forall j \in [m], |C \cap \mathcal{S}_j| > \frac{t|\mathcal{S}_j|}{n}$. It follows $\sum_{j\in[m]} |C \cap \mathcal{S}_j| > t$. However, we also have $\sum_{j\in[m]} |C \cap \mathcal{S}_j| \leq |C| \leq t$, which contradicts our assumption. □

The ratio-preservation of deterministic grouping introduces an avenue for more efficient IC and DKG. As we sketched in the introduction, both IC and DKG can be efficiently realized via a group of representatives, among whom at least one is honest. With deterministic grouping ensuring an honest majority in at least one group, we have the opportunity to treat each group as a “representative" to emulate the efficient representative-based protocols, thereby reducing communication.

In Section.4.1, we first introduce and construct a new broadcast primitive termed by *consortium-sender byzantine broadcast* (CSBB), which allows a group to broadcast an ℓ-bit message to the whole population at the cost of $O(n\ell + n^2\lambda)$, exactly matching the optimal cost of a single-sender broadcast. Then, in Section 4.2, we present efficient IC by leveraging CSBB together with deterministic grouping.

```
CSBB⟨$\mathcal{D}(\mathtt{msg}), \mathcal{P}$⟩
Init($1^\lambda, n$):
  • $(\mathsf{Sig}.vk_i, \mathsf{Sig}.sk_i) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$, generate signing keys for every $P_i \in \mathcal{P}$
  • $\mathsf{Acc.Gen}(1^\lambda, n) \rightarrow ak$ // publish the accumulator key as a CRS.
Round 1: each $D_i \in \mathcal{D}$ do
  • $(\mathsf{c}_1, \dots \mathsf{c}_n) \leftarrow \mathsf{Encode}(\mathtt{msg})$
  • $(u, \{w_j\}_{j\in[n]}) \leftarrow \mathsf{Accumulate}(ak, (\mathsf{c}_1, \dots, \mathsf{c}_n))$
  • send $(u, \mathsf{c}_j, w_j)$ to $P_j \in \mathcal{P}, \forall j \in [n]$
Round 2: each $P_j \in \mathcal{P}$ do
  • receive: $\{(u^{(i)}, \mathsf{c}_j^{(i)}, w_j^{(i)})\}$ from all $D_i$ in round 1
  • if $\exists I \subset [\eta] \wedge |I| \geq \eta - \tau$, s.t. $(u^{(i)}, \mathsf{c}_j^{(i)}, w_j^{(i)}) = (u^*, \mathsf{c}_j^*, w_j^*)$ for $\forall i \in I$ then
      – send $(\mathsf{c}_j^*, w_j^*)$ to all $P_{j'} \in \mathcal{P}$, and store $u^*$
  • else send $\perp$ to all $P_{j'} \in \mathcal{P}$, and store $u^* = \perp$
End of Round 2: each $P_j \in \mathcal{P}$ do
  • receive: $\{(\mathsf{c}_{j'}^*, w_{j'}^*)\}$ from all $P_{j'}$ in round 2
  • if $u^* = \perp$, then $\mathtt{msg}_j = \perp$
  • else set $C = \emptyset$
      – for $j' \in [n]$ do
          * if $\mathsf{Acc.Vrfy}(ak, u^*, (\mathsf{c}_{j'}^*), w_{j'}^*) = 1$, then $C = C \cup \{(j', \mathsf{c}_{j'}^*)\}$
          * if $|C| = t + 1$, then $\mathtt{msg}_j \leftarrow \mathsf{EC.Decode}(C)$, break
Round 3 to 2+$\Delta_{BA}$: // All participants $P_j \in \mathcal{P}$ run a BA to decide the output message
  • $\mathsf{BA}\langle P_j(\mathtt{msg}_j)\rangle \rightarrow \mathtt{msg}$
  • each $P_j$ outputs $\mathtt{msg}$.
```

Fig. 2. The CSBB protocol. $\Delta_{\mathsf{BA}(n)}$ denotes the number of rounds needed for the BA protocol for n participants. Σ is the signature scheme for authenticating messages.

4.1 Consortium-Sender Byzantine Broadcast

Definition. An (n, η, t, τ, ℓ) Consortium-Sender Byzantine Broadcast (CSBB) involves n participants $\mathcal{P} = \{P_1, \dots, P_n\}$ that has a subset $\mathcal{D} = \{D_1, \dots, D_\eta\} \subset \mathcal{P}$ acting as the sender consortium. The honest senders have the same ℓ-bit input $\mathtt{msg}$. We denote an instance of CSBB by $\mathsf{CSBB}\langle\mathcal{D}(\mathtt{msg}), \mathcal{P}\rangle$. A CSBB is secure if it satisfies the following properties against any PPT adversary $\mathcal{A}$ corrupting up to t parties in $\mathcal{P}$. (1) *Validity.* If all honest nodes in $\mathcal{D}$ have the same valid input $\mathtt{msg}$, and the adversary corrupts up to τ parties in $\mathcal{D}$, all honest parties in $\mathcal{P}$ output $\mathtt{msg}$.(2) *Agreement.* All honest parties output the same message.(3) *Termination.* All honest parties output a message.

A CSBB may have an initialization phase $\mathsf{Init}(1^\lambda, n)$ for PKI setup and generating CRS. We require the CSBB to retain all security properties even when there are polynomial many instances running *concurrently* in the unique identifier model (cf. Def.3.1) after the same initialization.

Building Blocks. We use a cryptographic accumulator Acc, an error correcting code EC, and a Byzantine agreement BA as building blocks. Particularly, Acc provides a succinct representation of a set while ensuring efficient membership verification. It incorporates the algorithms Gen for accumulator key generation, Eval to accumulate a set $\mathcal{S}$ into a value u, Wit to generate a witness w_i for an element $s_i \in \mathcal{S}$, and Vrfy to verify if s_i is in the set represented by u using w_i. EC includes deterministic algorithms Encode, which encodes a message into n code blocks $(\mathsf{c}_i)_{i\in[n]}$, and Decode to reconstruct a message from any $t+1$ code blocks. Formal definitions are recalled in Sect. 3.3.

Constructing CSBB. We give a construction for CSBB in Fig.2. A typical construction for BB is through the multicast-then-BA paradigm: BA guarantees all receivers output the same value. We follow a similar approach for CSBB while making necessary changes to the multicast phase to keep it efficient. Particularly, multicasting a value of ℓ bits to a population of n nodes incurs $O(n\ell)$ communication cost; if all η senders in the consortium perform the multicast, the cost will be $O(\eta n\ell)$, not better than independently invoking BB for η times when $\ell = O(n\lambda)$. To reduce the communication cost, we utilize the erasure code [13], which is a common trick in distributed protocols. More specifically, we let each sender in the consortium deterministically encode the $O(n\lambda)$-sized transcript into n blocks $(\mathsf{c}_1, \cdots, \mathsf{c}_n)$ each having $O(\lambda)$ bits, and send each c_i to P_i. P_i should receive multiple copies of c_i from the senders. However, it only multicasts the block, which appears most frequently to all other receivers. By doing so, the communication cost in the phase becomes $O(n\ell)$ again. When the sender consortium has an honest majority, P_i will only relay the correct block of the message. We also follow recent works to use a cryptographic accumulator [50] to help decode the erasure code in the presence of up to $n/2$ malicious blocks, such that the receiver should reconstruct the correct message sent by the sender consortium.

Analysis. We analyze the performance and security of our CSBB. At round 1, each sender in $\mathcal{D}$ sends out (u, c_j, w_j) whose size is $O(|\mathsf{w}| + \ell/n)$ to every $P_j \in \mathcal{P}$. The communication cost of this round is $O(\eta(\ell + n|\mathsf{w}|))$. At round 2, the cost is $O(n(\ell + n|\mathsf{w}|))$. Adding them together with the cost of BA, we have the total cost of $O((n+\eta)(\ell + n|\mathsf{w}|)) + \mathsf{BA}_n(\ell)$. Regarding computation, each sender needs to generate n witness, and each receiver needs to verify $O(n)$ witnesses w.r.t. Acc. We assume, without loss of generality, that the per-node computation cost is $O(n)$ group operations.

Regarding security, at a high level, *agreement* and *termination* are derived directly from the BA protocol. For *validity*, each P_j in the second round will yield the code c_j corresponding to that input. As a consequence, all P_j participants in the third round can reconstruct the initial input message. By the validity of BA, these participants should output the original input. The concurrent security essentially follows our intuition that honest nodes can ignore messages with different identifiers. Formally, we have the following theorem.

Theorem 4.2. *The protocol in Fig.2 is a concurrently secure (n, η, t, τ, ℓ)-CSBB for any $t < \frac{n}{2}$ and $\tau < \frac{\eta}{2}$ in the unique identifier model, assuming the underlying accumulator is secure, and the BA is concurrently secure in the unique identifier model against adaptive corruption of up to t nodes.*

First, we will give a general definition of concurrent security in the unique identifier model. Intuitively, a protocol could achieve concurrent security in the unique identifier model if an instance can aptly "ignore" messages with different identifiers, preserving its security as in the standalone

```
IC⟨{$P_i(msg_i\}_{i\in[n]}$⟩
Let $\{\mathcal{S}_1, \cdots, \mathcal{S}_m\}$ be a equal-sized partition over $[n]$
For any $P_i$: $vector_i = \{\bot, \cdots, \bot\}$, where $|vector_i| = n$ and $vector_i[j] = \bot$ for $j \in [n]$
Round 1: each $P_i \in \mathcal{P}$ do
  • Signature$(msg_i, sk_i) \to \sigma_i$; send $(msg_i, \sigma_i)$ to $P_j \in \mathcal{P}, \forall j \in [n]$
Round 2 to $\Delta_{\mathrm{BB}(\sqrt{n})} + 1$: each $P_i \in \mathcal{P}$ and $i \in \mathcal{S}_k$ do
  • if receive $(msg_i, \sigma_i)$ from any $P_j \in \mathcal{P}$ in round 1 and Verify$(msg_j, \sigma_j, pk_j) = 1$
    – $vector_i[j] = (msg_j, \sigma_j)$
  • broadcast: BB⟨$P_i(vector_i), \{P_j\}_{j\in\mathcal{S}_k}$⟩   // $P_i$ broadcasts the set $vector_i$ to $\{P_j\}_{j\in\mathcal{S}_k}$ via BB
End of Round $\Delta_{\mathrm{BB}(\sqrt{n})} + 1$: each $P_i \in \mathcal{P}$ and $i \in \mathcal{S}_k$ do
  • for any $j \in \mathcal{S}_k$ do
    – if receive $vector_j$ from $P_j$ in round 2
      * Update$(vector_i, vector_j)$   // update $vector_i$
Round $\Delta_{\mathrm{BB}(\sqrt{n})} + 2$ to $\Delta_{\mathrm{BB}(\sqrt{n})} + 1 + \Delta_{\mathrm{CSBB}(\sqrt{n},n)}$: for any $k \in [m]$ do
  • CSBB⟨$\{P_i(vector_i)\}_{i\in\mathcal{S}_k}, \mathcal{P}$⟩   // $\{P_i\}_{i\in\mathcal{S}_k}$ broadcasts the set $vector_k$ to $\mathcal{P}$ via CSBB
End of Round $\Delta_{\mathrm{BB}(\sqrt{n})} + 1 + \Delta_{\mathrm{CSBB}(\sqrt{n},n)}$: each $P_i \in \mathcal{P}$ do
  • set $vector_i[j] = \bot$ for $j \in [n]$
  • for $k \in [m]$ do   // receive message S from all $\mathcal{D}$
    – if receive $vector'_k$ from CSBB⟨$\{P_i\}_{i\in\mathcal{S}_k}, \mathcal{P}$⟩ in round 2+$\Delta_{BB}$+$\Delta_{CSBB}$
    – Update$(vector_i, vector'_k)$   // update $vector_i$
  • each $P_i$ outputs $vector_i$.
Function : Update$(vector_i, vector_j)$
  • for $s \in [n]$: if $vector_i[s] \neq \bot$ and $vector_j[s] \neq \bot$
    – if $vector_i[s] \neq vector_j[s]$ then $vector_i[s] = fault$
  • for $s \in [n]$: if $vector_i[s] = \bot$ and $vector_j[s] \neq bot$ and parse $vector_j[s] = (msg_s, \sigma_s)$
    – if Verify$(msg_s, \sigma_s, pk_s) = 1$ do : $vector_i[s] = vector_j[s]$
```

Fig. 3. The IC protocol. $\Delta_{\mathrm{BB}(n)}$ and $\Delta_{\mathrm{CSBB}(n)}$ represent the number of rounds required for the BB and CSBB protocols for n participants, respectively. σ denotes the signature utilized for authenticating messages.

setting. Though a message sent by P_i in an instance with identifier id can typically be crafted with access to the signing oracle $\mathsf{Sign}_{id}(sk_i, \cdot)$, we formalize the intuition as security against cross-instance signing queries (or CIS-Security).

Definition 4.3 (CIS security). Protocol Π uses signature scheme Σ. In its variant Π_{id}, it employs Σ_{id}. If Π_{id} retains its security properties for any id, even when facing adversaries with signing oracles $\mathsf{Sign}_{id'}(sk_i, \cdot)$ for any other id' not prefixed by id and any party's key sk_i, it's termed CIS-secure.

Now, we show our CSBB is *concurrently secure* in the unique identifier model. We start with the simple fact that our CSBB transcripts are perfectly simulatable with the help of a signing oracle since honest parties do not use any private input beyond the signing keys during the protocol execution.

Lemma 4.4. *Let $\mathcal{A}$ be a PPT adversary that corrupts an arbitrary number of nodes in $\mathcal{D}$ and $\mathcal{P}$ in an instance of CSBB with identifier id after* $\mathsf{Init}(1^\lambda, n)$. *There is a simulator* $\mathsf{S}^{\mathcal{A}}((\mathsf{Sig}.vk_i)_{i\in[n]})$ *with access to signing oracles* $\mathsf{Sign}_{id}(\mathsf{Sig}.sk_i, \cdot)$ *for any* $i \in [n]$, *outputting the view* $\mathsf{sview}_{\mathcal{A}}$ *whose distribution is identical to the distribution of the view of $\mathcal{A}$ in a real execution of this instance.*

Proof. We let the simulator $\mathsf{S}^{\mathcal{A}}$ run CSBB with the adversary $\mathcal{A}$ by acting on behalf of all honest nodes. Specifically, $\mathsf{S}^{\mathcal{A}}$ follows the protocol specification to generate all protocol messages, which is feasible because no message in our CSBB protocol requires secret input. Before sending a message $\mathtt{msg}$ on behalf of an honest node P_i, $\mathsf{S}^{\mathcal{A}}$ queries the oracle $\mathsf{Sign}_{id}(\mathsf{Sig}.sk_i, \cdot)$ with $\mathtt{msg}$, and then sends out $\mathtt{msg}$ along with the signature. At the point of $\mathcal{A}$'s view, the execution simulated by $\mathsf{S}^{\mathcal{A}}$ is identical to the real execution, and thus the distribution of the simulated view and that of the real view should be identical. □

Then, we present a generic result that shows CIS-security (cf. Def. 4.3) will imply concurrent security when the protocol transcripts can be perfectly simulatable by using access to signing oracles. It can be seen as a generalization to Lindell *et al.*'s result on BA protocols [45].

Lemma 4.5. *Let Π be a protocol that uses a signature scheme Σ in a black box manner. Let Π_{id} be a protocol which is obtained by replacing Σ with Σ_{id}. If for any PPT adversary $\mathcal{A}$, its view in an execution of Π_{id} can be perfectly simulated by a simulator $\mathsf{S}^{\mathcal{A}}$ with access to signing oracles $\mathsf{Sign}_{id}(\mathsf{Sig}.sk_i, \cdot)$, and Π satisfies the CIS security, then Π maintains its security even when polynomially many instances are executed concurrently in the unique identifier model.*

Proof. Assuming there are m instances of Π running concurrently with prefix-free identifiers $\{id_1, \ldots, id_m\}$, we argue the j-th instance maintains all the security properties, for an arbitrary $j \in [m]$. Since Π is CIS secure, Π_{id_j} shall be secure against any PPT adversary having access to signing oracles $\mathsf{Sign}_{id_{j'}}()$ for $j' \neq j$. However, if there exists a PPT adversary $\mathcal{A}$ that involves in all the m instances of Π and breaks the security of the j-th instance, we have a PPT adversary $\mathcal{B}$ with access to signing oracles $\mathsf{Sign}_{id_{j'}}()$ for $j' \neq j$ breaking the security of Π_{id_j}. The strategy of $\mathcal{B}$ is simple: it invokes $\mathcal{A}$ as a subroutine, forwards all messages between $\mathcal{A}$ and honest parties in Π_{id_j}, and simulates all other instances by using the signing oracles. From $\mathcal{A}$'s point of view, the environment simulated by $\mathcal{B}$ is identical to that of a real execution. Therefore, $\mathcal{B}$ can break the security of Π_{id_j} if $\mathcal{A}$ can break the security of j-th instance in the concurrent execution, which contradicts the CIS security. □

By Lemma.4.5 and 4.4, it would be sufficient for showing its concurrent security by showing its CIS-security.

Theorem 4.6. *The protocol in Fig.2 is a CIS-secure (n, η, t, τ, ℓ)-CSBB for any $t < \frac{n}{2}$ and $\tau < \frac{\eta}{2}$, assuming the underlying accumulator scheme is secure, and the BA protocol is CIS-secure against adversary corrupting up to t nodes. Moreover, the communication complexity of the CSBB is*

$$O((n + \eta) \cdot (\ell + n \cdot |w|)) + BA_n(\ell),$$

where $|w|$ is the size of a membership witness in the accumulator scheme, and $BA_n(\ell)$ is the communication of BA *among n participants with ℓ-bit inputs.*

Proof. Termination and agreement follow directly from the underlying BA protocol. To see validity, we analyze the status after each round, where an adversary $\mathcal{A}$ corrupts up tp τ parties in $\mathcal{D}$ and up to t parties in $\mathcal{P}$, and there are $\eta - \tau$ honest parties having the same input $\mathtt{msg}$. At the end of round 1, every honest P_j will receive the same $(u^*, \mathsf{c}_j^*, w_j^*)$ from the at least $\eta - \tau$ honest parties in $\mathcal{D}$, as Encode and $\mathsf{Accumulate}$ are deterministic, and thus will relay (c_j^*, w_j^*) to all other $P_i \in \mathcal{P}$. At the end of round 2, honest P_j receives $\{(\mathsf{c}_i^*, w_i^*)\}$ from other P_i's which contains at least $n - t$ honest pairs that pass the verification. On the other hand, by the unforgeability of the accumulator, if $\mathsf{Acc.Vrfy}(ak, u^*, (i, \mathsf{c}_i^*), w_i^*) = 1$, c_i^* must the the correct code of $\mathtt{msg}$ at i-th position. Therefore, every honest party P_j should reconstruct the same message $\mathtt{msg}$. Then, by the validity of the underlying BA protocol, all honest parties output $\mathtt{msg}$.

Regarding CIS security, we let honest parties ignore any message that is invalid under the current identifier. The CIS security of CSBB then follows the CIS security of BA and the fact that an adversary cannot forge a signature under the current identifier by leveraging signing oracles under other identifiers.

We then analyze its communication cost. At round 1, each sender in $\mathcal{D}$ sends out (u, c_j, w_j) whose size is $O(|\mathsf{w}| + \ell/n)$ to every $P_j \in \mathcal{P}$. The communication cost of this round is $O(\eta(\ell + n|\mathsf{w}|))$. At round 2, the cost is $O(n(\ell + n|\mathsf{w}|))$. Adding them together with the cost of BA, we have the total cost of $O((n+\eta)(\ell + n|\mathsf{w}|)) + \mathsf{BA}_n(\ell)$. □

4.2 Interactive Consistency

A direct application of CSBB yields an IC protocol with sub-cubic communication, as depicted in Figure 3. In this protocol, all honest nodes first send messages along with their signatures to all nodes. Subsequently, each node broadcasts the received inputs to other nodes within the same group, such that all nodes within the same group agree on the same input vector. This vector is then used as input for the CSBB protocol, guaranteeing that at least one vector is output to all nodes by the end of round $3 + \Delta_{BB} + \Delta_{CSBB}$. Consequently, all honest nodes output the same vector. We establish the following theorem.

Theorem 4.7. *Assuming that the underlying digital signature is secure, and BB and CSBB are concurrently secure in the unique identifier model, the protocol in Fig.3 is an adaptively secure (n, t, ℓ)-IC for any $t < \frac{n}{2}$,*

Proof. Firstly, the termination can be directly derived from the underlying BB and CSBB protocols. Secondly, the agreement is guaranteed by the following factors: 1) The CSBB protocol ensures that all honest nodes produce identical outputs, as they participate in the same multiple instances of CSBB, resulting in consistent outcomes. 2) At the end of round $3 + \Delta_{BB} + \Delta_{CSBB}$, all honest nodes initial an empty set vector. 3) Furthermore, the Update procedure is deterministic. Taken together, these factors result in identical outputs for all honest nodes. Thirdly, based on the following fact, the validity condition is trivially satisfied: the security of the digital signature guarantees the unforgeability of the signature. Moreover, given that at least one partition constitutes the honest majority, there exists at least one CSBB output set that includes the inputs from all honest nodes. Therefore, the final output must contain the inputs from all honest nodes. □

5 SIMULATION-BASED DEFINITIONS FOR PVSS

In this section, we provide simulation-based definitions for publicly verifiable secret sharing (PVSS), making it a valuable tool in the realm of DKG. With the new definitions, we will be able to analyze our DKGs in a modular way.

Brief Overview. 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. PVSS takes VSS a step further: all encrypted shares, complete with verification proof, are placed on a public channel for universal validation. An *aggregatable* PVSS can compress multiple PVSS transcripts into one single publicly verifiable transcript.

The Need for Simulation-based Definitions. One of DKG's core goals is to act as a stand-in for the trusted key generation phase of threshold cryptosystems. Given this, DKG should be able to emulate standard key generation to serve a wide range of distributed cryptography applications. Thus, DKG usually uses simulation-based security modeling.

However, PVSS was mostly formulated using a game-based indistinguishability definition, termed IND-secrecy [19]. This definition doesn't fully capture the essence of key distribution from an adversary's perspective. It overlooks potential malleability challenges, which, in a DKG setting, could allow adversaries to arbitrarily sway key distribution. This makes a security reduction for DKG using PVSS as a black box inherently challenging.

To bridge this gap, we put forth simulation-based definitions for PVSS. Notably, we enrich PVSS's syntax to include the set of creator identities (CID) in each transcript. This prevents adversaries from merely replicating a simulated transcript. It's worth mentioning that recent research by Bacho and Loss [9] also formalized aggregatable PVSS and incorporated the ID into its syntax. However, their primary application was to randomness beacons, and their definitions did not adopt a simulation-based approach.

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 they differ by the size of their CID set), 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.
- $\mathsf{Deal}((ek_i)_{i\in[n]}, \mathtt{cid}) \to (\mathsf{Trans}, sk)$: 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 the CID set $\{\mathtt{cid}\}$.
- $\mathsf{Agg}(\{(\mathsf{Trans}_i)\}_{i\in[m]}, (ek_i)_{i\in[n]}) \to \mathsf{Trans}$: It outputs an aggregated transcript Trans whose CID set is $\{\mathtt{cid}_i\}_{i\in[m]}$, where $\mathtt{cid}_i$ is from Trans_i.
- $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}) \to b$: It checks if Trans is valid.
- $\mathsf{getCID}(\mathsf{Trans}) \to \{\mathtt{cid}_i\}_{i\in[m]}$: It returns the CID set.
- $\mathsf{PubDriv}(\mathsf{Trans}) \to (pk, (pk_i)_{i\in[n]})$: 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 $|\mathbb{I}| = t+1$.

Security. A PVSS scheme should satisfy *correctness*, *soundness*, *secrecy*, and *simulation soundness*. In the following definitions, we use $\mathsf{Init}_{\mathcal{A}}(1^\lambda, n) \to (\mathtt{crs}, C, (ek_i, dk_i)_{i\notin C}, (ek_i)_{i\in C}, st_{\mathcal{A}})$ to denote an execution of the initial phase involving the adversary $\mathcal{A}$, where C is the set of corrupted nodes, and $st_{\mathcal{A}}$ is the state of the adversary.

- **Correctness:** Assume $\mathsf{Init}_{\mathcal{A}}(1^\lambda, n)$ has been done. For $(\mathsf{Trans}, sk) \leftarrow \mathsf{Deal}(ek_1, ek_2, \ldots, ek_n)$, the transcript can always be verified, *i.e.*,
$$\Pr[\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}) = 1] = 1, \tag{1}$$
Assume $\{\mathsf{Trans}_j\}_{j\in[m]}$ are valid "native" transcripts, and $\mathsf{PubDriv}(\mathsf{Trans_j}) \to (pk^{(j)}, (pk_i^{(j)}))$. For $\mathsf{Agg}(\{\mathsf{Trans}_j\}_{j\in[m]}, (ek_i)_{i\in[n]}) \to \mathsf{Trans}$, it holds that $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}) = 1$, and
$$\mathsf{PubDriv}(\mathsf{Trans}) = (\bigotimes_{j\in[m]} pk^{(j)}, (\bigotimes_{j\in[m]} pk_i^{(j)})_{i\in[n]}).$$
- **Soundness:** Any adversary cannot produce a valid transcript while it will be decrypted to a set of inconsistent shares. Formally, assume $\mathsf{Init}_{\mathcal{A}}(1^\lambda, n)$ has been done. If a transcript is verified, *i.e.*, $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}) = 1$, then for any two subsets $\mathbb{I}_1$ and $\mathbb{I}_2$ of $t+1$ uncorrupted participants, the secret recovered from the transcript is unique, *i.e.*,
$$\Pr\left[\mathsf{Rec}(\{\mathsf{Dec}(ek_i, dk_i, \mathsf{Trans})\}_{i\in\mathbb{I}_1}) = \mathsf{Rec}(\{\mathsf{Dec}(ek_i, dk_i, \mathsf{Trans})\}_{i\in\mathbb{I}_2})\right] = 1. \tag{2}$$

- **Secrecy:** There is a triple of PPT simulators $\{\mathsf{SInit}, \mathsf{SDeal}, \mathsf{SRec}\}$. Such that, for any static PPT adversary $\mathcal{A}$ which corrupts up to t nodes, a PVSS transcript by an honest dealer does not leak sk beyond its public information, *i.e.*, for any $\mathtt{cid}$,

$$\left| \Pr\left[\begin{array}{l} \mathsf{Init}_{\mathcal{A}}(1^\lambda, n) \to (\mathtt{crs}, C, (ek_i, dk_i)_{i\notin C}, (ek_i)_{i\in C}, st_{\mathcal{A}}) \\ \mathsf{Deal}((ek_i)_{i\in[n]}, \mathtt{cid}) \to (\mathsf{Trans}, sk) : \\ \quad \mathcal{A}(\mathtt{crs}, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathsf{Trans}) = 1 \end{array}\right] - \Pr\left[\begin{array}{l} \mathsf{KeyGen}(1^\lambda) \to (pk, sk), \\ \mathsf{SInit}_{\mathcal{A}}(1^\lambda, n) \to (\mathtt{crs}, C, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathtt{tk}), \\ \mathsf{SDeal}((ek_i)_{i\in[n]}, pk, \mathtt{tk}, \mathtt{cid}) \to \mathsf{Trans} : \\ \quad \mathcal{A}(\mathtt{crs}, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathsf{Trans}) = 1 \\ \quad \wedge\ \mathsf{PubDriv}(\mathsf{Trans}) = (pk, \cdot). \end{array}\right] \right| \leq \mathsf{negl}(\lambda), \tag{3}$$

  where KeyGen is the default key generation algorithm of $(\mathcal{PK}, \mathcal{SK})$.

- **Simulation soundness.** Some form of soundness must be preserved, even after the adversary sees a simulated transcript. Formally, for any static PPT adversary, it holds that

$$\Pr\left[\begin{array}{l} \mathsf{KeyGen}(1^\lambda) \to (pk, sk), \\ \mathsf{SInit}_{\mathcal{A}}(1^\lambda, n) \to (\mathtt{crs}, C, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathtt{tk}), \\ \mathsf{SDeal}((ek_i)_{i\in[n]}, pk, \mathtt{tk}, \mathtt{cid}) \to \mathsf{Trans}, \\ \mathcal{A}(\mathtt{crs}, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathsf{Trans}) \to \mathsf{Trans}^* \\ \mathsf{SRec}(\mathtt{tk}, \mathsf{Trans}^*) \to sk^*, \mathsf{PubDriv}(\mathsf{Trans}^*) \to (pk^*, \cdot) : \\ \quad (pk^*, sk^*) \in \mathsf{Rela} \wedge \mathtt{cid} \notin \mathsf{getCID}(\mathsf{Trans}^*) \\ \quad \wedge\ \mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}^*) = 1 \end{array}\right] \leq \mathsf{negl}(\lambda). \tag{4}$$

  We need both soundness and simulation soundness. The former does not directly imply the latter due to different ways of extraction.

PVSS instantiation. Conventional PVSS schemes [33, 37] with minor enhancements can meet our definitions. In Appendix.A, we present a secure PVSS scheme for the standard key structure in DLog-based cryptography, i.e., $sk \in \mathbb{Z}_p$ and $pk = g^{sk} \in \mathbb{G}$. This scheme is obtained by following the general "encrypt-and-proof" paradigm and additionally applying a signature of knowledge (SoK) to embed a creator ID into its transcript. We further explicitly employ Scrape's technique [19] to improve its verification time. While the PVSS scheme in Appendix.A could have various instantiations, we summarize the result about an LWE-based instantiation (which can be seen as a variant of [37]) as its concrete performance stands out.

Lemma 5.1. *Under the LWE assumption and DL assumption [37], there is a PVSS scheme for the standard key structure in DLog-based cryptography, satisfying correctness, soundness, secrecy, and simulation soundness. Particularly, the transcript size of* $\mathtt{Trans}$ *is* $O((n)\lambda)$. *Both* Deal *and* PubVrfy *require* $O(n)$ *group operations. The computational costs for other functions are minor, approximately* $O(1)$ *group operations.*

Aggregatable PVSS instantiation. All known aggregatable PVSS are variants of Scrape PVSS [19, 41] for the pairing-based key structure. For completeness, we present a variant of Scrape PVSS in Appendix.B, which meets our definitions. We summarize the result of the instantiation in the following lemma.

Lemma 5.2. *Under the SXDH and BDG assumption [41], there is an aggregatable PVSS scheme for the pairing-based key structure, satisfying correctness, soundness, secrecy, and simulation soundness. Particularly, the transcript size of* $\texttt{Trans}$ *is* $O((n+m)\lambda)$*, where* m *represents the number of transcripts aggregated into* $\texttt{Trans}$*. Both* Deal *and* $\mathsf{PubVrfy}$ *require* $O(n)$ *group operations, whereas* Agg *demands* $O(n \log m)$ *group operations. The computational costs for other functions are minor, approximately* $O(1)$ *group operations.*

6 DRAGON-DKG

In this section, we introduce a new secret sharing paradigm called *Consortium-Dealer Secret Sharing* (CDSS) to realize the idea of DRAGON-DKG. We then outline a DKG framework built from CDSS and present a CDSS construction for better DKG.

6.1 Consortium-Dealer Secret Sharing: Definition

We formalize CDSS, which enables a consortium of dealers to distribute shares of a *random value* to a large population. Particularly, for DKG, we let CDSS also return a public key of the shared secret along with a list of public key shares that correspond to secret shares obtained by each receiver.

Syntax. An (n, η, t, τ)-CDSS scheme for $(\mathcal{PK}, \mathcal{SK})$ involves n participants $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ with a special subset $\mathcal{D} = \{D_1, D_2, \ldots, D_\eta\} \subset \mathcal{P}$ acting as a dealer consortium. It consists of an initialization phase Init, a deal protocol $\mathsf{Deal}\langle\mathcal{D}, \mathcal{P}\rangle$, and a reconstruction algorithm Rec.

(1) $\mathsf{Init}(1^\lambda, n)$. This sets up the PKI and generates a CRS $\texttt{crs}$. (2) $\mathsf{Deal}\langle\mathcal{D}, \mathcal{P}\rangle$. At the end of the protocol, each receiver P_i outputs a public key $pk \in \mathcal{PK}$, a sequence of public key shares $(pk_i)_{i\in[n]} \in \mathcal{PK}^n$, and a secret key share $sk_i \in \mathcal{SK}$. (3) $\mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}})$. It reconstructs the secret key sk for pk. We require Rec to be linear, i.e., it first determines coefficients $\{\alpha_i\}_{i\in\mathbb{I}}$ where $\alpha_i \in \mathbb{N}$ and reconstructs the secret $sk = \bigoplus_{i\in\mathbb{I}} \alpha_i sk_i$ for any subset $\mathbb{I} \in [n]$ with $|\mathbb{I}| = t + 1$.

We consider the *robustness* and *key-expressibility* of CDSS in the *multi-instance* setting. Assume that after an honest initialization phase $\mathsf{Init}(1^\lambda, n)$, there are m instances $\{\mathsf{Deal}\langle\mathcal{D}^{(j)}, \mathcal{P}\rangle\}_{j\in[m]}$ running concurrently in the unique identifier model (cf. Def. 3.1). Assume there is a PPT adversary $\mathcal{A}$ that corrupts $\{P_i\}_{i\in C}$ for $|C| \le t$. We detail each property below.

Multi-instance robustness. For any $\mathcal{A}$ and any integer m polynomial in λ, we have the following guarantees for each instance: (1) For the j-th instance where $|\{P_i\}_{i\in C} \cap \mathcal{D}^{(j)}| \le \tau$, all honest P_i's output properly. That is, every P_i outputs the same public tuple $(pk, (pk_i)_{i\in[n]})$ and its secret share sk_i such that $\mathsf{Rela}(pk_i, sk_i) = 1$. Meanwhile, for any two subsets $\mathbb{I}_1, \mathbb{I}_2 \subset [n]$ and $|\mathbb{I}_1| = |\mathbb{I}_2| = t + 1$, it follows that the same sk is reconstructed from $\{sk_i\}_{i\in\mathbb{I}_1}$ and $\{sk_i\}_{i\in\mathbb{I}_2}$, and $(pk, sk) \in \mathsf{Rela}$. (2) For the j-th instance where $|\{P_i\}_{i\in C} \cap \mathcal{D}^{(j)}| > \tau$, all honest receivers outputs (or $\perp$).

Multi-instance key-expressibility. For any $\mathcal{A}$ and any integer m polynomial in λ such that $\exists j \in [m], |\{P_i\}_{i\in C} \cap \mathcal{D}^{(j)}| \le \tau$, there is a PPT simulator algorithm $\mathsf{Sim}^{\mathcal{A}}$. For any PPT distinguisher $\mathcal{A}'$, it holds that

$$\left| \Pr\left[\begin{array}{l} \langle\{\mathsf{Deal}^{\mathcal{A}}_{id_j}\langle\mathcal{D}^{(j)}, \mathcal{P}\rangle\}_{j\in[m]}\rangle \to ((\mathsf{out}_j)_{j\in[m]}, \mathsf{view}_{\mathcal{A}}), \\ s.t.\ \mathsf{out}_j = pk^{(j)} \textbf{ or } \perp\colon \mathcal{A}'((\mathsf{out}_j)_{j\in[m]}, \mathsf{view}_{\mathcal{A}}) = 1 \end{array}\right] - \Pr\left[\begin{array}{c} \mathsf{KeyGen}(1^\lambda) \to (pk, sk), \mathsf{Sim}^{\mathcal{A}}(pk) \to (\{\mathsf{tup}\}_{j\in[m]}, \\ \mathsf{sview}_{\mathcal{A}}), s.t.\ \mathsf{tup}_j = (sk'^{(j)}, pk'^{(j)}, \alpha^{(j)}) \textbf{ or } \perp, \\ \mathsf{out}_j \leftarrow \alpha^{(j)} \cdot pk \otimes pk'^{(j)}, \textbf{ or } \mathsf{out}_j \leftarrow \perp \textbf{ if } \mathsf{tup}_j = \perp\colon \\ (\textbf{if } \mathsf{tup}_j \ne \perp, \textbf{ then } (sk'^{(j)}, pk'^{(j)}) \in \mathsf{Rela}) \\ \wedge\ (\exists j^*, \alpha^{(j^*)} \ne 0) \wedge \mathcal{A}'((\mathsf{out}_j)_{j\in[m]}, \mathsf{sview}_{\mathcal{A}}) = 1 \end{array}\right] \right| \le \mathsf{negl}(\lambda).$$

$\langle\{\mathsf{Deal}^{\mathcal{A}}_{id_j}\langle\mathcal{D}^{(j)},\mathcal{P}\rangle\}_{j\in[m]}\rangle \rightarrow ((\mathtt{out}_j)_{j\in[m]},\mathsf{view}_{\mathcal{A}})$ denotes a concurrent execution of m CDSS instances $\{\mathsf{Deal}_{id_j}\}_{j\in[m]}$ involving $\mathcal{A}$ following an execution of the same initialization. $\mathtt{out}_j$ is the public key $pk^{(j)}$ that an honest P_i outputs in the instance or $\perp$ if it aborts; $\mathsf{view}_{\mathcal{A}}$ is the view of the adversary $\mathcal{A}$, including all public messages and its internal states. KeyGen is the default key generation algorithm for $(\mathcal{PK},\mathcal{SK})$.

6.2 CDSS for Group-Element Secret

Intuition. A straightforward yet non-succinct construction could let every dealer in the consortium $\mathcal{D}$ run a complete secret sharing to all receivers. To reduce the communication cost, our idea is to let the receiver receive one “aggregated" and valid secret share instead of sending multiple shares to be aggregated. In particular, we leverage the aggregatable PVSS (see definitions in Sect.5 and the construction in Appendix.B), which enables us to delegate the share aggregation without harming the secrecy. Then, the dealer consortium may broadcast the aggregated PVSS transcript via CSBB.

The construction. Notably, the dealer consortium needs to agree on one aggregated PVSS before broadcasting it via our CSBB. For secrecy, it is crucial to ensure that the secret w.r.t. the aggregated PVSS remains unknown to an adversary, which may corrupt τ out of η dealers and t out of n receivers. We guarantee the secrecy by letting each dealer generate an (n,t)-PVSS transcript under the public keys of receivers, broadcast to the *dealer consortium*, and then aggregate all valid PVSS transcripts. In particular, note that the final secret key is the sum of secret keys shared by all dealers, which means the adversary cannot know the final secret key unless it corrupts all dealers. Meanwhile, by the definition of PVSS, an adversary corrupting t receivers cannot learn the secret key from the decrypted shares. Moreover, as each dealer sends its PVSS transcript via a BB protocol, it ensures all dealers have the same view of valid transcripts and thus obtain the same aggregated transcript.

Formally, assume an aggregatable PVSS for $(\mathcal{PK},\mathcal{SK})$, a Byzantine Broadcast protocol BB, and a CSBB protocol CSBB. We delineate the deal protocol of CDSS in Fig.4, while its reconstruction algorithm is the same as PVSS.Rec. The initialization algorithm $\mathsf{Init}(1^\lambda,n)$ is as follows: It invokes $\mathsf{CSBB.Init}(1^\lambda,n)$, which includes a PKI setup for a digital signature scheme Σ, and $\mathsf{PVSS.Init}(1^\lambda,n)$ which generates $(ek_i,dk_i)_{i\in[n]}$ for the PVSS scheme.

Communication complexity. We first analyze the communication complexity of our CDSS construction in Fig.4. All η dealers broadcast a PVSS transcript of size $O(n\lambda)$ bits, which incurs bit complexity of $\eta\mathsf{BB}_\eta(n\lambda)$ in total, and dealers and receivers invoke a CSBB protocol to disseminate the aggregated PVSS transcript, which incurs bit complexity of $O(n^2\cdot\mathsf{w})+\mathsf{BA}_n(n\lambda)$, assuming $\eta<n$ and the witness size w. The communication complexity of CDSS is

$$O(n^2\cdot\mathsf{w})+\eta\mathsf{BB}_\eta(n\lambda)+\mathsf{BA}_n(n\lambda), \tag{5}$$

where $\mathsf{BB}_z(\ell)$ (or $\mathsf{BA}_z(\ell)$) is the communication cost of Byzantine Broadcast BB (or Byzantien Agreement BA) among z participants with ℓ-bit input.

Computation complexity. In our design, each dealer creates one PVSS transcript, verifies η transcripts, and aggregates η transcripts; each receiver verifies one transcript. They invoke CSBB once, costing $O(n)$ group operations per node. With the PVSS scheme in Appendix.B, the per-node computation cost is $O(n)$.

On the complexity of the DKG. The bit communication complexity of the DKG (Fig.6) is equal to m (the number of shards) times the complexity of the CDSS construction. Therefore, with CDSS in Fig.4, the bit communication complexity of our DKG is $O(mn^2\cdot|\mathsf{w}|)+n\mathsf{BB}_\eta(n\lambda)+m\mathsf{BA}_n(\lambda)$, while $n=\eta m$.

Now, we discuss the best sharding parameters for the smallest communication. Assuming we are using the optimal BA and BB, *i.e.*, $\mathsf{BA}_z(\ell)=\mathsf{BB}_z(\ell)=O(z\ell+z^2\lambda)$, and the accumulator with witness

```
Deal⟨$\mathcal{D}, \mathcal{P}$⟩
Round 1 to $\Delta_{\mathsf{BB}(\eta)}$: each $D_j \in \mathcal{D}$ do
  • PVSS.Deal($(ek_i)_{i\in[n]}, \mathtt{cid}_{D_j}$) $\to$ ($\mathtt{Trans}_j, sk_j$)
  • broadcast: BB⟨$D_j(\mathtt{Trans}_j), \mathcal{D}$⟩ with an unique identifier $id_j^{\mathsf{BB}}$
End of Round $\Delta_{\mathsf{BB}(\eta)}$: each $D_j \in \mathcal{D}$ do
  • receive broadcast messages $\{\mathtt{Trans}_{j'}\}_{j'\in[\eta]}$ from all $D_{j'} \in \mathcal{D}$
  • set TRANS = $\emptyset$
  • for $j' \in [\eta]$
    – if PVSS.PubVrfy($(ek_i)_{i\in[n]}, \mathtt{Trans}_{j'}$) = 1 and PVSS.getCID($\mathtt{Trans}_{j'}$) = $\{\mathtt{cid}_{D_{j'}}\}$ then
      * TRANS = TRANS $\cup$ $\{\mathtt{Trans}_{j'}\}$
  • PVSS.Agg(TRANS, $(ek_i)_{i\in[n]}$) $\to$ $\mathtt{Trans}$
Round $\Delta_{\mathsf{BB}(\eta)} + 1$ to $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{CSBB}(\eta,n)}$:
  • CSBB⟨$\mathcal{D}(\mathtt{Trans}), \mathcal{P}$⟩   // The dealer consortium $\mathcal{D}$ broadcasts the Trans to $\mathcal{P}$ via CSBB
End of Round $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{CSBB}(\eta,n)}$: each $P_i$ do
  • receive broadcast message $\mathtt{Trans}$ from all $\mathcal{D}$, and let CID $\leftarrow$ PVSS.getCID($\mathtt{Trans}$)
  • if PVSS.PubVerify($(ek_i)_{i\in[n]}, \mathtt{Trans}$) = 0 $\vee \neg$(CID $\subset \{\mathtt{cid}_{D_j}\}_{j\in[\eta]} \wedge |\mathsf{CID}| \geq \tau + 1$) then
    – output $\perp$
  • else $(pk, pk_1, \ldots, pk_n) \leftarrow$ PVSS.PubDerive($\mathtt{Trans}$), $sk_i \leftarrow$ PVSS.Dec($ek_i, dk_i, \mathtt{Trans}$)
    – output $(pk, pk_1, \ldots, pk_n, sk_i)$
```

Fig. 4. Deal⟨$\mathcal{D}, \mathcal{P}$⟩ Protocol of complete CDSS. $\Delta_{\mathsf{BB}(\eta)}$ (or $\Delta_{\mathsf{CSBB}(\eta,n)}$) is the number of rounds needed for running BB with n parties (or CSBB with η senders and n receivers). We assume every P_i has its publicly known CID, denoted by $\mathtt{cid}_{P_i}$.

size $|\mathsf{w}| = O(\lambda)$, we notice that $\eta = m = \sqrt{n}$ yields a communication cost of DKG which is $O(n^{2.5}\lambda)$. Regarding computation cost, with the PVSS in Appendix.B, the per-node computation cost of the DKG is $O(n^{1.5})$ group operations.

Security analysis. We establish the security of our CDSS in the following.

Theorem 6.1. *Assuming that the underlying* PVSS *is secure, and BB and CSBB are concurrently secure in the unique identifier model, the* CDSS *protocol in Fig.4 satisfies the multi-instance robustness and the multi-instance key-expressibility.*

We prove the multi-instance robustness in Lemma.6.2 and the multi-instance key-expressibility in Lemma.6.3

Lemma 6.2. *The* CDSS *protocol satisfies multi-instance robustness, assuming concurrent security of BB and CSBB, and correctness and soundness of PVSS.*

Proof. First, we argue that for any instance j, all honest nodes either return $\perp$ or output properly, *i.e.*, they have the same view of public information $(pk^{(j)}, (pk_i)_{i\in[n]}^{(j)})$ and obtain a correct secret share, despite there is a PPT adversary $\mathcal{A}$ corrupting $\{P_i\}_{i\in C}$ for $|C| \leq t$. By agreement of CSBB, all honest receivers P_i's will receive the same message and will return $\perp$ if the message is not a valid PVSS transcript or its CID is not consistent with its dealer consortium. Then, by the soundness of PVSS, when the PVSS transcript is valid, every honest receiver can obtain the correct secret share by decrypting the transcript.

Then, we show that for the instance j where $|\{P_i\}_{i\in C} \cap \mathcal{D}^{(j)}| \leq \tau$, the honest parties must output properly. Since at most τ nodes in $\mathcal{D}^{(j)}$ are corrupted, the BB instances within $\mathcal{D}$ have all the security guarantees. By the agreement of BB, all honest nodes in $\mathcal{D}$ will receive the message from each instance of BB. Moreover, by the correctness of PVSS and the validity of BB, an honest node in $\mathcal{D}$ broadcasts a valid PVSS transcript that will be received by all honest nodes in $\mathcal{D}$. Therefore, all honest nodes in $\mathcal{D}$ can receive at least $\eta - \tau$ valid PVSS transcripts and can aggregate them into one. By the *validity* of CSBB, all nodes in $\mathcal{P}$ receive the same valid transcript. Then, by the soundness of PVSS, all P_i's can obtain a valid secret share and derive the same public key (shares). □

Lemma 6.3. *The* CDSS *protocol satisfies the multi-instance key-expressibility, assuming the concurrent security of BB and CSBB, and the correctness, soundness, secrecy, and simulation soundness of PVSS.*

Proof. We proceed with this proof by constructing the simulator algorithm $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ which leverages the simulator algorithms $\{\mathsf{PVSS.SInit}, \mathsf{PVSS.SDeal}, \mathsf{PVSS.SRec}\}$ of the PVSS. For clarity, we write down the pseudo-code of $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ below, which takes as input pk which is generated by the default key generation algorithm $\mathsf{KeyGen}(1^\lambda) \longrightarrow (pk, sk)$.

```
$\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}(pk)$
Initialize: $\mathsf{CSBB.Init}_{\mathcal{A}}(1^\lambda, n)$ and $\mathsf{PVSS.SInit}_{\mathcal{A}}(1^\lambda, n) \longrightarrow (\mathtt{crs}, C, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathtt{tk})$
Run the $m$ instances$\{\mathsf{Deal}^{\mathcal{A}}_{id_j}\}_{j\in[m]}$ with $\mathcal{A}$, as below:
Select $j^* \in [m]$ and $i^* \in [\eta]$ s.t. $|\mathcal{D}^{(j^*)} \cap \{P_i\}_{i\in C}| \le \tau$, and $D^{(j^*)}_{i^*} \in \mathcal{D}^{(j^*)} \setminus \{P_i\}_{i\in C}$
Round 1 to $\Delta_{\mathsf{BB}(\eta)}$:
  • Run $\mathsf{SDeal}((ek_i)_{i\in[n]}, pk, \mathtt{tk}, D^{(j^*)}_{i^*}) \longrightarrow \mathsf{Trans}^*, \mathsf{BB}\langle D^{(j^*)}_{i^*}(\mathsf{simTrans}), \mathcal{D}^{(j^*)}\rangle$
  • Honestly execute the code for every $D^{(j)}_i \notin \{P_i\}_{i\in C}$ and $(i, j) \neq (i^*, j^*)$
Honestly execute the remaining rounds, besides performing the tasks below:
At the end of round $\Delta_{\mathsf{BB}(\eta)}$:
  • Let $\mathsf{TRANS}^*$ be the set of valid PVSS transcripts in $\mathcal{D}^{(j^*)}$
  • for $\mathsf{Trans}_i \in \mathsf{TRANS}^*$ and $\mathsf{Trans}_i \neq \mathsf{Trans}^*$
      – $\mathsf{PVSS.SRec}(\mathtt{tk}, \mathsf{Trans}_i) \longrightarrow sk'_i$, and $\mathsf{PVSS.PubDriv}(\mathsf{Trans}_i) \longrightarrow pk'_i$
  • Let $pk'^{(j^*)} = \bigotimes pk'_i, sk'^{(j^*)} = \bigoplus sk'_i, \alpha^{(j^*)} = 1$
At the end of round $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{CSBB}(\eta,n)} + 1$:
  • for $j \in [m]$ and $j \neq j^*$
      – Receive $\mathsf{Trans}^{(j)}$ from CSBB by $\mathcal{D}^{(j)}, \forall j \in [m]; \mathsf{CID}^{(j)} \longleftarrow \mathsf{PVSS.getCID}(\mathsf{Trans}^{(j)})$
      – if $\mathsf{PVSS.PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}^{(j)}) = 1 \wedge |\mathsf{CID}^{(j)} \subset \{\mathtt{cid}_{D^{(j)}_i}\}_{i\in[\eta]}| \ge \tau + 1$ then
          * $\mathsf{PVSS.SRec}(\mathtt{tk}, \mathsf{Trans}^{(j)}) \longrightarrow sk'^{(j)}; \mathsf{PVSS.PubDriv} \longrightarrow (pk'^{(j)}, \cdot)$
          * $\alpha^{(j)} = 0; \mathtt{tup}_j = (sk'^{(j)}, pk'^{(j)}, \alpha^{(j)})$
        else $\mathtt{tup}_j = \perp$
return $(\{\mathtt{tup}_j\}_{j\in[m]}, \mathsf{view}_{\mathcal{A}})$
```

Recall the multi-instance key-expressibility definition, the PPT distinguisher $\mathcal{A}'$ is required to distinguish $(\{\alpha^{(j)} \cdot pk \otimes pk'^{(j)}\}_{j\in\mathbb{J}'}, \mathsf{view}_{\mathcal{A}})$ (provided by the above simulator) and $(\{pk^{(j)}\}_{j\in\mathbb{J}}, \mathsf{view}_{\mathcal{A}})$ (from a real execution), where $\mathbb{J}' = \{j \in [m] : \mathsf{Trans}_j \text{ is valid}\}$ in the simulated execution and $\mathbb{J} = \{j \in [m], \mathsf{Trans}_j \text{ is valid}\}$ in the real execution. Note that $\{\alpha^{(j)} \cdot pk \otimes pk'^{(j)} = pk^{(j)}\}_{j\in\mathbb{J}'}$ can be derived from $\mathcal{A}$'s view. At the point of $\mathcal{A}$'s view, the only difference between the execution simulated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ and the real execution is about how $\mathsf{PVSS.Init}$ is executed, and how Trans^* is generated. By the secrecy of PVSS, the distinguisher $\mathcal{A}'$ cannot distinguish the two tuples with a non-negligible advantage. Note that the CID of $\mathsf{Trans}^{(j)}$ does not contain $\mathtt{cid}_{D^{(j^*)}_{i^*}}$ for $j \neq j^*$. By the *simulation* *soundness* of PVSS, PVSS.SRec can obtain $sk^{(j)}$ which is the valid secret key of $pk^{(j)}$. Moreover, we have $\alpha = \alpha^{(j^*)} = 1$. $\square$

```
                              Deal⟨$\mathcal{D}, \mathcal{P}$⟩
Round 1 to $\Delta_{\mathsf{DKG}(\eta)}$:   //$\mathcal{D}$ run a DKG protocol, and each $D_i$ obtains a secret key share $sk_{\mathcal{D},i}$
  • $\mathsf{DKG}^{\mathsf{unp}}\langle\mathcal{D}\rangle \rightarrow (pk_{\mathcal{D}}, sk_{\mathcal{D},1}, \ldots, sk_{\mathcal{D},\eta})$
Round 1 +$\Delta_{\mathsf{DKG}(\eta)}$ to $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)}$: each $D_j \in \mathcal{D}$ do
  • $\mathsf{PVSS.Deal}((ek_i)_{i\in[n]}, \mathtt{cid}_{D_j}) \rightarrow (\mathtt{Trans}_j, sk_j)$
  • broadcast: $\mathsf{BB}\langle D_j(\mathtt{Trans}_j), \mathcal{D}\rangle$ with an unique identifier $id_j^{\mathsf{BB}}$
Round $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)} + 1$ to $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)} + 2$: each $D_i \in \mathcal{D}$ do
  • receive broadcast messages $\{\mathtt{Trans}_{j'}\}_{j'\in[\eta]}$ from all $D_{j'} \in \mathcal{D}$
  • set $\mathsf{TRANS} = \emptyset$
  • for $j' \in [\eta]$
      – if $\mathsf{PVSS.PubVrfy}((ek_i)_{i\in[n]}, \mathtt{Trans}_{j'}) = 1$ and $\mathsf{PVSS.getCID}(\mathtt{Trans}_{j'}) = \{\mathtt{cid}_{D_{j'}}\}$ then
          * $\mathsf{TRANS} = \mathsf{TRANS} \cup \{\mathtt{Trans}_{j'}\}$
  • send $sk_{\mathcal{D},i}$ to all $D_j \in \mathcal{D}$
End of Round $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)} + 2$:
  • receive secret shares $\{sk_{\mathcal{D},j}\}$, reconstruct $sk_{\mathcal{D}}$, and compute $\mathsf{Hash}(sk_{\mathcal{D}}) \rightarrow (j_1, \ldots, j_\kappa)$
  • set $\mathsf{OutTRANS} = \{\mathsf{Trans}_j \in \mathsf{TRANS} : j \in \{j_1, \ldots, j_\kappa\}\}$
Round $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)} + 3$ to $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)} + \Delta_{\mathsf{CSBB}(\eta,n)} + 2$:
  • $\mathsf{CSBB}\langle\mathcal{D}(\mathsf{OutTRANS}), \mathcal{P}\rangle$   // $\mathcal{D}$ broadcasts the selected transcript set OutTRANS to $\mathcal{P}$ via CSBB
End of Round $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)} + \Delta_{\mathsf{CSBB}(\eta,n)} + 2$: each $P_i$ do
  • receive message OutTRANS from all $\mathcal{D}$, and parse $\mathsf{OutTRANS} = (\mathtt{Trans}_{j_1}, \ldots, \mathtt{Trans}_{j_\kappa})$
  • for $\mathtt{Trans}_{j_z} \in \mathsf{OutTRANS}$
      – if $\mathsf{PVSS.PubVerify}((ek_i)_{i\in[n]}, \mathtt{Trans}_{j_z}) = 0 \vee \mathsf{PVSS.getCID}(\mathtt{Trans}_{j_z}) \not\subset \{\mathtt{cid}_{D_j}\}_{j\in[\eta]}$ then
          * output $\perp$
  • for $\mathtt{Trans}_{j_z} \in \mathsf{OutTRANS}$ do
      – $(pk^{(j_z)}, pk_1^{(j_z)}, \ldots, pk_n^{(j_z)}) \leftarrow \mathsf{PVSS.PubDerive}(\mathtt{Trans}_{j_z})$
      – $sk_i^{(j_z)} \leftarrow \mathsf{PVSS.Dec}(ek_i, dk_i, \mathtt{Trans}_{j_z})$
  • output $(pk = \bigotimes_{i\in[\kappa]} pk^{(j_z)}, pk_1 = \bigotimes_{i\in[\kappa]} pk_1^{(j_z)}, \ldots, pk_n = \bigotimes_{i\in[\kappa]} pk_n^{(j_z)}, sk_i = \bigoplus_{i\in[\kappa]} sk_i^{(j_z)})$
```

Fig. 5. Deal$\langle\mathcal{D}, \mathcal{P}\rangle$ Protocol of CDSS for field-element secrets. $\Delta_{\mathsf{BB}(\eta)}$ (or $\Delta_{\mathsf{CSBB}(\eta,n)}$, or $\Delta_{\mathsf{DKG}(\eta)}$) is the number of rounds need for running BB with n parties (or CSBB with η senders and n receivers, or DKG with η parties). We assume every P_i has its publicly known CID, denoted by $\mathtt{cid}_{P_i}$.

6.3 CDSS for Field-Element Secret

In DLog-based cryptography, most conventional cryptographic schemes possess a secret key within the finite field $\mathbb{Z}_p$. However, the aggregatable PVSS used in CDSS above is primarily aligned with group-element secrets (unless using generic zkSNARK[38]). Now we turn to present a CDSS construction that can be built upon a conventional PVSS without aggregation (see the candidate construction in Appendix A), which is compatible with field-element secrets.

Note that the aggregatable PVSS in the above CDSS is employed to reduce communication while maintaining security. Here, we introduce a different path for achieving these goals by utilizing a common coin "within" the dealer consortium. For example, in the group-element CDSS, all dealers first broadcast their PVSS transcripts within the consortium. However, after this step, the dealers do not aggregate these valid transcripts; Instead, they invoke a common coin [12] to randomly pick κ PVSS transcripts, which will be sent out together via the CSBB protocol. Here, κ is the statistic security parameter that is independent of n or η. A receiver can obtain its share by decrypting all the κ transcripts and adding the decrypted values together. With a high probability $(1-(\frac{\tau}{\eta})^{\kappa})$, at least one of the selected transcripts is from an honest dealer, which guarantees the secrecy of this scheme.

Formally, assume a PVSS for $(\mathcal{PK}, \mathcal{SK})$, a BB, a CSBB, a DKG protocol $\mathsf{DKG}^{\mathsf{unp}}$ which we only assume its secret key is unpredictable, and a hash function Hash which maps $\{0,1\}^*$ to κ indexes in the range of $[\eta]$. We elucidate the deal phase of CDSS scheme in Fig.5, while its reconstruction algorithm is the same as $\mathsf{PVSS.Rec}$. The initialization algorithm $\mathsf{Init}(1^\lambda, n)$ is as follows: It invokes $\mathsf{CSBB.Init}(1^\lambda, n)$, which includes a PKI setup for a signature scheme Σ, $\mathsf{PVSS.Init}(1^\lambda, n)$ which generates $(ek_i, dk_i)_{i\in[n]}$ for the PVSS scheme, and the setup for $\mathsf{DKG}^{\mathsf{unp}}$.

Performance analysis. With the PVSS scheme outlined in Lemma 5.1, the PVSS transcript size is $O(n\lambda)$. Therefore, the communication cost incurred by all η dealers broadcasting their PVSS transcripts within the consortium is $\eta\mathsf{BB}_\eta(n\lambda)$. The communication cost for broadcasting κ PVSS transcripts via CSBB is $O(n^2\lambda\kappa\cdot|w|)+\mathsf{BA}_n(n\lambda\kappa)$, where $|w|$ is size of an accumulator witness. With optimal BA/BB and an accumulator with a constant-sized witness, the above communication cost will be $O(n^2\lambda\kappa)$. While the dealer consortium needs to perform a DKG within the consortium, the communication cost for the DKG is at most $O(\eta^3\lambda)$ without using the sub-cubic DKG in Sect.6, which is not the dominating term for either communication or computation cost. For computation cost, since each receiver needs to process κ transcripts, the per-node computation cost is $O(n\kappa)$ group operations.

Security analysis. We establish the security of the CDSS scheme in Fig.5 in the following.

Theorem 6.4. *Assuming the underlying* PVSS *is secure, the* $\mathsf{DKG}^{\mathsf{unp}}$ *is unpredictable and robust, and* BB *and* CSBB *are concurrently secure in the unique identifier model, the* CDSS *in Fig.5 satisfies the multi-instance robustness and the multi-instance key-expressibility in the random oracle model.*

The proof largely resembles to the proof for Theorem 6.1, except that we need to leverage the fact that the selected κ transcripts include one from an honest dealer. We discuss the multi-instance robustness in Lemma 6.5 and the multi-instance key-expressibility in Lemma 6.6, respectively.

Lemma 6.5. *The* CDSS *protocol satisfies multi-instance robustness in the random oracle model, assuming concurrent security of BB and CSBB, the correctness and soundness of PVSS, and that the* $\mathsf{DKG}^{\mathsf{unp}}$ *is unpredictable and robust.*

Proof. The agreement of CSBB ensures that all honest receivers P_i's will receive the same message, which could be a PVSS transcript or $\perp$. The soundness of PVSS ensures that when the PVSS transcript is valid, every honest receiver can obtain the correct share by decrypting the transcript. Therefore, we have that for any instance j, all honest nodes either return $\perp$ or output properly.

We then argue that for the instance j where $|\{P_j\}_{j\in C}\cap\mathcal{D}^{(j)}|\leq\tau$, the honest parties output properly. By the robustness of $\mathsf{DKG}^{\mathsf{unp}}$, the set of dealers $\mathcal{D}^{(j)}$ will obtain a set of consistent secret shares at the end of Round $\Delta_{\mathsf{DKG}(\eta)}$. By the unpredictability of $\mathsf{DKG}^{\mathsf{unp}}$, the output of Hash is distinguishable with κ uniformly sampled indexes from $[\eta]$. Therefore, the probability of no honest dealer being selected is bounded by $(\frac{\tau}{\eta})^{\kappa}+\mathsf{negl}(\lambda)$, which is negligibly small. By the validity of BB, the selected honest node should have correctly broadcasted a valid PVSS to the dealer consortium. It follows that the dealer consortium will at least broadcast one valid PVSS transcript, such that all receivers must output properly for the dealer consortium. □

[12]Particularly, we can implement the coin protocol by letting the consortium first run a DKG protocol, then reconstruct the secret key, and finally apply a hash function (which is modeled as a random oracle) to the secret key.

Lemma 6.6. *The CDSS protocol satisfies multi-instance key-expressibility in the random oracle model, assuming the concurrent security of BB and CSBB, the correctness, soundness, secrecy, and simulation soundness of PVSS, and the robustness and unpredictability of* $\mathsf{DKG}^{\mathsf{unp}}$.

Proof. We proceed with this proof by constructing the simulator algorithm $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ which leverages the simulator algorithms $\{\mathsf{PVSS.SInit}, \mathsf{PVSS.SDeal}, \mathsf{PVSS.SRec}\}$ of the PVSS. For clarity, we write down the pseudo-code of $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ below, which takes as input pk which is generated by the default key generation algorithm $\mathsf{KeyGen}(1^\lambda) \to (pk, sk)$.

```
$\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}(pk)$
**Initialize:** $\mathsf{CSBB.Init}_{\mathcal{A}}(1^\lambda, n)$, $\mathsf{PVSS.SInit}_{\mathcal{A}}(1^\lambda, n) \to (\mathtt{crs}, C, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathtt{tk})$, and setup for $\mathsf{DKG}^{\mathsf{unp}}$
Run the $m$ instances$\{\mathsf{Deal}^{\mathcal{A}}_{id_j}\}_{j\in[m]}$ with $\mathcal{A}$, as below:
**Round 1 to $\Delta_{\mathsf{DKG}(\eta)}$:**
  • Honestly execute the DKG protocols on behalf of honest nodes
**Round 1 + $\Delta_{\mathsf{DKG}(\eta)}$ to $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{DKG}(\eta)}$:**
  • Select $j^* \in [m]$ *s.t.* $|\mathcal{D}^{(j^*)} \cap \{P_i\}_{i\in C}| \le \tau$
  • Reconstruct $sk_{\mathcal{D}^{(j^*)}}$ using the honest parties's shares, and compute $\mathsf{Hash}(sk_{\mathcal{D}^{(j^*)}}) \to \{j_1, \ldots, j_\kappa\}$
  • Select $i^* \in \{j_1, \ldots, j_\kappa\}$ *s.t.* $D^{(j^*)}_{i^*} \in \mathcal{D}^{(j^*)} \setminus \{P_i\}_{i\in C}$      // It can find such $i^*$ w.h.p.
**Round 1 to $\Delta_{\mathsf{BB}(\eta)}$:**
  • Run $\mathsf{SDeal}((ek_i)_{i\in[n]}, pk, \mathtt{tk}, D^{(j^*)}_{i^*}) \to \mathsf{Trans}^*$, $\mathsf{BB}\langle D^{(j^*)}_{i^*}(\mathsf{simTrans}), \mathcal{D}^{(j^*)}\rangle$
  • Honestly execute the code for every $D^{(j)}_i \notin \{P_i\}_{i\in C}$ and $(i, j) \ne (i^*, j^*)$
Honestly execute the remaining rounds, besides performing the tasks below:
**At the end of round $\Delta_{\mathsf{BB}(\eta)}$:**
  • Let $\mathsf{TRANS}^*$ be the set of valid PVSS transcripts produced by $D_{j_1}, \ldots, D_{j_\kappa} \in \mathcal{D}^{(j^*)}$
  • for $\mathsf{Trans}_i \in \mathsf{TRANS}^*$ and $\mathsf{Trans}_i \ne \mathsf{Trans}^*$
    – $\mathsf{PVSS.SRec}(\mathtt{tk}, \mathsf{Trans}_i) \to sk'_i$, and $\mathsf{PVSS.PubDriv}(\mathsf{Trans}_i) \to pk'_i$
  • Let $pk'^{(j^*)} = \bigotimes pk'_i$, $sk'^{(j^*)} = \bigoplus sk'_i$, $\alpha^{(j^*)} = 1$
**At the end of round $\Delta_{\mathsf{BB}(\eta)} + \Delta_{\mathsf{CSBB}(\eta,n)} + 1$:**
  • for $j \in [m]$ and $j \ne j^*$
    – Receive $\mathsf{OutTrans}^{(j)}$ from CSBB by $\mathcal{D}^{(j)}$; parse $\mathsf{OutTrans}^{(j)} = \{\mathtt{Trans}^{(j)}_{i_1}, \ldots, \mathtt{Trans}^{(j)}_{i_z}\}$
    – for $i' \in \{i_1, \ldots, i_z\}$
      * if $\mathsf{PVSS.PubVrfy}((ek_i)_{i\in[n]}, \mathsf{Trans}^{(j)}_{i'}) = 0 \lor \mathsf{PVSS.getCID}(\mathsf{Trans}^{(j)}_{i'}) \notin \{\mathtt{cid}_{D^{(j)}_i}\}_{i\in[\eta]}$
          · then $\mathtt{tup}_j = \perp$; continue
      * else for $\forall\, a \in [z]$ :
          · $\mathsf{PVSS.SRec}(\mathtt{tk}, \mathsf{Trans}^{(j)}_{i_a}) \to sk'^{(j)}_a$; $\mathsf{PVSS.PubDriv} \to (pk'^{(j)}_a, \cdot)$; $\alpha^{(j)}_a = 0$
          · $\mathtt{tup}_j = (\bigoplus_{a\in[z]} sk'^{(j)}_a,\ \bigotimes_{a\in[z]} pk'^{(j)}_a,\ \alpha^{(j)} = \sum_{a\in[z]} \alpha^{(j)}_a)$
**return** $(\{\mathtt{tup}_j\}_{j\in[m]}, \mathsf{view}_{\mathcal{A}})$
```

Recall the multi-instance key-expressibility definition, the PPT distinguisher $\mathcal{A}'$ is required to distinguish $(\{\alpha^{(j)} \cdot pk \otimes pk'^{(j)}\}_{j\in\mathbb{J}'}, \mathsf{view}_{\mathcal{A}})$ (provided by the above simulator) and $(\{pk^{(j)}\}_{j\in\mathbb{J}}, \mathsf{view}_{\mathcal{A}})$ (from a real execution), where $\mathbb{J}' = \{j \in [m] : \mathsf{Trans}_j \text{ is valid}\}$ in the simulated execution and $\mathbb{J} = \{j \in [m], \mathsf{Trans}_j \text{ is valid}\}$ in the real execution. By the robustness and unpredictability of $\mathsf{DKG}^{\mathsf{unp}}$, the simulator $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ can finish the above simulation with an overwhelming probability. The following arguments are similar to those for Lemma 6.3. Particularly, under the condition that the simulator could finish the above simulation, at the point of $\mathcal{A}$'s view, the only difference between the execution simulated by $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ and the real execution is about how PVSS.Init is executed, and how Trans* is generated. By the secrecy of PVSS, the distinguisher $\mathcal{A}'$ cannot distinguish the two tuples with a non-negligible advantage. Note that the CID of $\mathsf{Trans}^{(j)}$ does not contain $\mathtt{cid}_{D_{i^*}^{(j^*)}}$ for $j \neq j^*$. By the *simulation soundness* of PVSS, PVSS.SRec can obtain $sk^{(j)}$ which is the valid secret key of $pk^{(j)}$. Moreover, we have $\alpha = \alpha^{(j^*)} = 1$. □

6.4 DKG from CDSS

Let $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ be the whole population. Let $\mathcal{S} = \{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_m\}$ be an arbitrary partition of $[n]$. W.l.o.g, we assume for every $\mathcal{S}_i$ and $\mathcal{S}_j$ it holds that $|S_i| = |S_j| = \eta = n/m$. Then, we can have a DKG for the key structure $\{\mathcal{PK}, \mathcal{SK}\}$ over $\mathcal{P}$, by parallelly invoking m CDSS instances, where each shard $\{P_i\}_{i\in\mathcal{S}_j}$ acts like a dealer consortium. We present the protocol description in Fig.6, while its initialization phase is CDSS.Init and its reconstruction algorithm is CDSS.Rec.

```
DKG protocol DKG⟨$\mathcal{P}$⟩ for $\{\mathcal{PK}, \mathcal{SK}\}$
let $\{\mathcal{S}_1, \cdots, \mathcal{S}_m\}$ be a equal-sized partition over [n]
Round 1 to $\Delta_{\mathsf{CDSS}(\frac{n}{m},n)}$:    // Each shard $\{P_i\}_{i\in\mathcal{S}_j}$ acts as a dealer consortium to distribute a secret
  • parallely run CDSS.Deal$\langle\{P_i\}_{i\in\mathcal{S}_j}, \mathcal{P}\rangle$ with unique identifiers $\forall j \in [m]$
At the end of round $\Delta_{\mathsf{CDSS}(\frac{n}{m},n)}$: each $P_i \in \mathcal{P}$ do
  • group $m$ CDSS intances into $\mathbb{J}_1, \mathbb{J}_2$ s.t., $\mathbb{J}_1 \cup \mathbb{J}_2 = [m]$, where $\mathbb{J}_1$: default ouputs $\perp$; $\mathbb{J}_2$: valid ouputs
    $\{pk^{(j)}, (pk_z^{(j)})_{z\in[n]}, sk_i^{(j)}\}_{j\in\mathbb{I}_2}$
  • output: $pk = \bigotimes_{j\in\mathbb{J}_2} pk^{(j)}$,   $(pk_z = \bigotimes_{j\in\mathbb{J}_2} pk_z^{(j)})_{z\in[n]}$,   $sk_i = \bigoplus_{j\in\mathbb{J}_2} sk_i^j$
```

Fig. 6. DKG from CDSS. $\Delta_{\mathsf{CDSS}(\eta,n)}$ is the number of rounds needed for running the CDSS's deal protocol CDSS.Deal with a dealer consortium of η members and a receiver set of n nodes.

The security of our DKG inherently stems from the multi-instance security of the CDSS, given the fact that at least one shard maintains an honest majority.

Theorem 6.7. *The DKG protocol in Fig.6 is a secure (n,t)-DKG, which satisfies robustness and key-expressibility against static adversaries, if the underlying* CDSS *is a secure (η,τ)-*CDSS *for some integer $\tau \geq \frac{\eta t}{n}$, satisfying multi-instance robustness and multi-instance key-expressibility.*

Proof. The robustness of the DKG is implied by the multi-instance robustness of the underlying CDSS. Assume that the adversary $\mathcal{A}$ corrupts $\{P_i\}_{i\in C}$ for $|C| = t$. By Lemma.4.1, there exists at least one $j^* \in [m]$, such that $|\{P_i\}_{i\in C} \cap \mathcal{D}^{(j^*)}| \leq \tau$. Then, by multi-instance robustness, all honest participants should output $\perp$ for all $j \in \mathbb{J}_1$ and output properly for all $j \in \mathbb{J}_2$; Moreover, $\mathbb{J}_2$ is non-empty, as $j^* \in \mathbb{J}_2$. For each $j \in \mathbb{J}_2$, we have $(pk_i^{(j)}, sk_i^{(j)}) \in \mathsf{Rela}$ for $i \in [n]$. By the homomorphism of Rela, it follows $(sk_i = \bigoplus_{j\in\mathbb{J}_2} sk_i^j, pk_i = \bigotimes_{j\in\mathbb{J}_2} pk_i^{(j)}) \in \mathsf{Rela}$. Then, we argue the secret key sk reconstructed from any subset of $t+1$ secret shares will be identical and satisfy $(pk, sk) \in \mathsf{Rela}$. For $\mathbb{I} \subset [n]$ s.t. $|\mathbb{I}| = t+1$, the reconstruction algorithm of CDSS will determine $\{\alpha_i\}_{i\in\mathbb{I}}$. Then, the unique secret key $sk^{(j)}$ of j-th CDSS is $\bigoplus_{i\in\mathbb{I}} \alpha_i sk_i^{(j)}$ for every $j \in \mathbb{J}_2$. By description, the reconstruction algorithm of DKG is also CDSS.Rec, which means the final secret key reconstructed from $\{(i, sk_i)\}_{i\in\mathbb{I}}$ is $sk = \bigoplus_{i\in\mathbb{I}_1} \alpha_i (\bigoplus_{j\in\mathbb{J}_2} sk_i^{(j)}) = \bigoplus_{j\in\mathbb{J}_2} sk^{(j)}$, which is determined by $\{sk^{(j)}\}$ and independent of $\mathbb{I}$.

The key-expressibility is implied by the multi-instance key-expressibility of the CDSS. We proceed the proof by constructing the simulator $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ for any PPT adversary $\mathcal{A}$ which corrupts $\{P_i\}_{i\in C}$ for some $|C| = t$. $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}$ directly invokes the simulator $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}$ and "aggregates" its outputs.

For clarity, we have written the code for the simulator below.

$\mathsf{Sim}^{\mathcal{A}}_{\mathsf{DKG}}(pk)$
Invoke $\mathsf{Sim}^{\mathcal{A}}_{\mathsf{CDSS}}(pk) \rightarrow (\{\mathtt{tup}\}_{j\in[m]}, \mathsf{sview}_{\mathcal{A}})$, *s.t.* $\mathtt{tup}_j = (sk'^{(j)}, pk'^{(j)}, \alpha^{(j)})$ **or** $\perp$,   Compute $sk' = \bigoplus_{j\in\mathbb{J}} sk'^{(j)}, pk' = \bigotimes_{j\in\mathbb{J}} pk'^{(j)}, \alpha = \sum_{j\in\mathbb{J}} \alpha^{(j)}$, for $\mathbb{J} = \{j : \mathtt{tup}_j \neq\perp\}$ **return** $(sk', pk', \alpha, \mathsf{sview}_{\mathcal{A}})$

By the definition of multi-instance key-expresability, no PPT distinguisher $\mathcal{A}''$ can distinguish $\{(\alpha^{(j)} \cdot pk \otimes pk'^{(j)})_{j\in\mathbb{J}}, \mathsf{sview}_{\mathcal{A}}\}$ and $((pk^{(j)})_{j\in\mathbb{J}_2}, \mathsf{view}_{\mathcal{A}})$ from the real execution. Notice that

$$\alpha \cdot pk \otimes pk' = \bigoplus_{j\in\mathbb{J}} (\alpha^{(j)} \cdot pk \otimes pk'^{(j)}) \text{ and } \bigoplus_{j\in\mathbb{J}_2} (pk^{(j)})$$

are obtained by applying the same operations on the two tuples. Therefore, there is no PPT distinguisher $\mathcal{A}'$ that can distinguish

$$(\alpha \cdot pk \otimes pk', \mathsf{sview}_{\mathcal{A}}) \text{ and } (\bigoplus_{j\in\mathbb{J}_2} (pk^{(j)}), \mathsf{view}_{\mathcal{A}})$$

with a non-negligible advantage. Thus, the DKG satisfies key-expressibility. □

Instantiating group-element DKG. The bit communication of the DKG (Fig.6) is equal to m (the number of shards) times the complexity of the CDSS construction. Therefore, with CDSS in Fig.4, the bit communication complexity of our DKG is $O(mn^2 \cdot |\mathsf{w}|) + n\mathsf{BB}_\eta(n\lambda) + m\mathsf{BA}_n(\lambda)$, while $n = \eta m$.

Now, we discuss the best sharding parameters for the smallest communication. Assuming we are using the optimal BA and BB, *i.e.*, $\mathsf{BA}_z(\ell) = \mathsf{BB}_z(\ell) = O(z\ell + z^2\lambda)$, and the accumulator with witness size $|\mathsf{w}| = O(\lambda)$, we notice that $\eta = m = \sqrt{n}$ yields a communication cost of DKG which is $O(n^{2.5}\lambda)$. Regarding computation cost, with the PVSS in Appendix.B, the per-node computation cost of the DKG is $O(n^{1.5})$ group operations.

Instantiating field-element DKG. As the CDSS in Fig.5 satisfies both multi-instance robustness and multi-instance key-expressibility, it can be plugged into the DKG construction in Fig.6. The security of the resulting DKG follows Theorem 6.7. Regarding performance, the DKG parallelly invokes $m = \sqrt{n}$ instances of CDSS, and thus the total communication complexity will be $O(n^{2.5}\lambda\kappa)$, and the per-node computation cost will be $O(n^{1.5}\kappa)$ group operations.

7 CONCLUSION

Both distributed key generation (DKG) and interactive consistency (IC) can be efficiently achieved with a small committee containing at least one honest representative. However, relying on common randomness to sample such a committee is costly, often necessitating DKG. In this work, we introduce a set of *Dragon* techniques, including consortium-sender Byzantine broadcast and consortium-dealer secret sharing, which enable the entire population, when arbitrarily grouped, to emulate such a committee in a decentralized manner. Our methods eliminate the reliance on common coins while retaining the efficiency benefits of committee-based approaches. Consequently, our developed DKG protocols (for both group-element and field-element secrets) and IC enjoy sub-cubic communication complexity.

An interesting theoretical question is whether we can have a DKG protocol with $O(n^2\lambda)$ communication complexity, matching the lower bound established in [56], or whether this lower bound is tight. Moreover, we believe our techniques and ideas have broader applications beyond DKG and IC, leaving further exploration as an interesting avenue for future work.

ACKNOWLEDGEMENT

This work was supported in part by research awards from Ethereum Foundation, Protocol Labs, Stellar Development Foundation, and SOAR Prize from the University of Sydney.

REFERENCES

[1] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2023. Communication complexity of byzantine agreement, revisited. *Distributed Comput.* 36, 1 (2023), 3–28.

[2] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023. Bingo: Adaptivity and Asynchrony in Verifiable Secret Sharing and Distributed Key Generation. In *CRYPTO (1) (Lecture Notes in Computer Science, Vol. 14081)*. Springer, 39–70.

[3] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. 2023. Communication and Round Efficient Parallel Broadcast Protocols. Cryptology ePrint Archive, Paper 2023/1172. https://eprint.iacr.org/2023/1172 https://eprint.iacr.org/2023/1172.

[4] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. 2023. Communication and Round Efficient Parallel Broadcast Protocols. Cryptology ePrint Archive, Paper 2023/1172. https://eprint.iacr.org/2023/1172 https://eprint.iacr.org/2023/1172.

[5] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. 2021. Compressed $\varSigma $-Protocols for Bilinear Group Arithmetic Circuits and Application to Logarithmic Transparent Threshold Signatures. In *ASIACRYPT (4) (Lecture Notes in Computer Science, Vol. 13093)*. Springer, 526–556.

[6] Sarah Azouvi and Marko Vukolic. 2022. Pikachu: Securing PoS Blockchains from Long-Range Attacks by Checkpointing into Bitcoin PoW using Taproot. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus*. 53–65.

[7] Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2023. *GRandLine: Adaptively Secure DKG and Randomness Beacon with (Almost) Quadratic Communication Complexity*. Technical Report. Cryptology ePrint Archive. https://eprint.iacr.org/2023/1887.

[8] Renas Bacho and Julian Loss. 2022. On the Adaptive Security of the Threshold BLS Signature Scheme. In *CCS*. ACM, 193–207.

[9] Renas Bacho and Julian Loss. 2023. Adaptively Secure (Aggregatable) PVSS and Application to Distributed Randomness Beacons. In *CCS*. ACM.

[10] Michael Ben-Or and Ran El-Yaniv. 2003. Resilient-optimal interactive consistency in constant time. *Distributed Comput.* 16, 4 (2003), 249–262.

[11] Adithya Bhat, Aniket Kate, Kartik Nayak, and Nibesh Shrestha. 2023. OptRand: Optimistically responsive distributed random beacons. In *NDSS*.

[12] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. 2021. RandPiper - Reconfiguration-Friendly Random Beacons with Quadratic Communication. In *CCS*. ACM, 3502–3524.

[13] Richard E Blahut. 1983. *Theory and practice of error control codes*. Addison-Wesley.

[14] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. 2015. *On Bitcoin as a public randomness source*. Technical Report. Cryptology ePrint Archive. https://eprint.iacr.org/2015/1015.

[15] Elizabeth Borowsky and Eli Gafni. 1993. Generalized FLP impossibility result for t-resilient asynchronous computations. In *STOC*. ACM, 91–100.

[16] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.

[17] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random oracles in constantipole: practical asynchronous Byzantine agreement using cryptography (extended abstract). In *PODC*. ACM, 123–132.

[18] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Adaptive Security for Threshold Cryptosystems. In *CRYPTO (Lecture Notes in Computer Science, Vol. 1666)*. Springer, 98–115.

[19] Ignacio Cascudo and Bernardo David. 2017. SCRAPE: Scalable Randomness Attested by Public Entities. In *ACNS (Lecture Notes in Computer Science, Vol. 10355)*. Springer, 537–556.

[20] Ignacio Cascudo and Bernardo David. 2020. ALBATROSS: Publicly AttestabLe BATched Randomness Based On Secret Sharing. In *ASIACRYPT (3) (Lecture Notes in Computer Science, Vol. 12493)*. Springer, 311–341.

[21] Ignacio Cascudo, Bernardo David, Lydia Garms, and Anders Konring. 2022. YOLO YOSO: Fast and Simple Encryption and Secret Sharing in the YOSO Model. In *ASIACRYPT (1) (Lecture Notes in Computer Science, Vol. 13791)*. Springer, 651–680.

[22] Melissa Chase and Anna Lysyanskaya. 2006. On Signatures of Knowledge. In *CRYPTO (Lecture Notes in Computer Science, Vol. 4117)*. Springer, 78–96.

[23] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *FOCS*. IEEE Computer Society, 383–395.

[24] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. 2001. Multiparty Computation from Threshold Homomorphic Encryption. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 2045)*. Springer, 280–299.

[25] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. 2022. Spurt: Scalable Distributed Randomness Beacon with Transparent Setup. In *SP*. IEEE, 2502–2517.

[26] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. 2023. *A New Paradigm for Verifiable Secret Sharing*. Technical Report. Cryptology ePrint Archive. https://eprint.iacr.org/2023/1196.

[27] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. In *SP*. IEEE, 2518–2534.

[28] Danny Dolev and Rüdiger Reischuk. 1982. Bounds on Information Exchange for Byzantine Agreement. In *PODC*. ACM, 132–140.

[29] Danny Dolev and H. Raymond Strong. 1983. Authenticated Algorithms for Byzantine Agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666.

[30] Paul Feldman. 1987. A Practical Scheme for Non-interactive Verifiable Secret Sharing. In *FOCS*. IEEE Computer Society, 427–437.

[31] Hanwen Feng, Tiancheng Mai, and Qiang Tang. 2023. *Scalable and Adaptively Secure Any-Trust Distributed Key Generation and All-hands Checkpointing*. Technical Report. Cryptology ePrint Archive. https://eprint.iacr.org/2023/1773.

[32] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM* 32, 2 (1985), 374–382.

[33] Pierre-Alain Fouque and Jacques Stern. 2001. One Round Threshold Discrete-Log Key Generation without Private Channels. In *Public Key Cryptography (Lecture Notes in Computer Science, Vol. 1992)*. Springer, 300–316.

[34] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Efficient Asynchronous Byzantine Agreement without Private Setups. In *ICDCS*. IEEE, 246–257.

[35] Rosario Gennaro, Steven Goldfeder, and Arvind Narayanan. 2016. Threshold-Optimal DSA/ECDSA Signatures and an Application to Bitcoin Wallet Security. In *ACNS (Lecture Notes in Computer Science, Vol. 9696)*. Springer, 156–174.

[36] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *J. Cryptol.* 20, 1 (2007), 51–83.

[37] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. 2022. Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties. In *EUROCRYPT (1) (Lecture Notes in Computer Science, Vol. 13275)*. Springer, 458–487.

[38] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *EUROCRYPT (2) (Lecture Notes in Computer Science, Vol. 9666)*. Springer, 305–326.

[39] Jens Groth and Victor Shoup. 2023. *Fast batched asynchronous distributed key generation*. Technical Report. Cryptology ePrint Archive. https://eprint.iacr.org/2023/1175.

[40] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. In *CCS*. ACM, 803–818.

[41] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Aggregatable Distributed Key Generation. In *EUROCRYPT (1) (Lecture Notes in Computer Science, Vol. 12696)*. Springer, 147–176.

[42] Damien Imbs, Michel Raynal, and Julien Stainer. 2016. Are Byzantine Failures Really Different from Crash Failures?. In *DISC (Lecture Notes in Computer Science, Vol. 9888)*. Springer, 215–229.

[43] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT (Lecture Notes in Computer Science, Vol. 6477)*. Springer, 177–194.

[44] Valerie King and Jared Saia. 2010. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In *PODC*. ACM, 420–429.

[45] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. 2002. On the composition of authenticated byzantine agreement. In *STOC*. ACM, 514–523.

[46] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC*. ACM, 129–138.

[47] Atsuki Momose and Ling Ren. 2021. Optimal Communication Complexity of Authenticated Byzantine Agreement. In *DISC (LIPIcs, Vol. 209)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 32:1–32:16.

[48] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *DISC (LIPIcs, Vol. 179)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:17.

[49] Wafa Neji, Kaouther Blibech Sinaoui, and Narjes Ben Rajeb. 2016. Distributed key generation protocol with a new complaint management strategy. *Secur. Commun. Networks* 9, 17 (2016), 4585–4595.

[50] Lan Nguyen. 2005. Accumulators from Bilinear Pairings and Applications. In *CT-RSA (Lecture Notes in Computer Science, Vol. 3376)*. Springer, 275–292.

[51] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT (Lecture Notes in Computer Science, Vol. 1592)*. Springer, 223–238.

[52] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (1980), 228–234.

[53] Torben P. Pedersen. 1991. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO (Lecture Notes in Computer Science, Vol. 576)*. Springer, 129–140.

[54] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO (Lecture Notes in Computer Science, Vol. 5157)*. Springer, 554–571.

[55] Victor Shoup. 2023. *The many faces of Schnorr.* Technical Report. Cryptology ePrint Archive. https://eprint.iacr.org/2023/1019.

[56] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. 2021. *Synchronous Distributed Key Generation without Broadcasts.* Technical Report. Cryptology ePrint Archive. https://eprint.iacr.org/2021/1635.

[57] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabien Laguillaumie, and Giulio Malavolta. 2021. Efficient CCA Timed Commitments in Class Groups. In *CCS*. ACM, 2663–2684.

[58] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. 2020. Towards Scalable Threshold Cryptosystems. In *IEEE Symposium on Security and Privacy*. IEEE, 877–893.

[59] Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. 2022. Gossiping for Communication-Efficient Broadcast. In *CRYPTO (3) (Lecture Notes in Computer Science, Vol. 13509)*. Springer, 439–469.

[60] Gang Wang and Mark Nixon. 2020. RandChain: Practical Scalable Decentralized Randomness Attested by Blockchain. In *Blockchain*. IEEE, 442–449.

[61] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. 2022. hbACSS: How to Robustly Share Many Secrets. In *NDSS*. The Internet Society.

[62] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling Blockchain via Full Sharding. In *CCS*. ACM, 931–948.

[63] Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. 2022. Polynomial Commitment with a One-to-Many Prover and Applications. In *USENIX Security Symposium*. USENIX Association, 2965–2982.

A A PVSS SCHEME WITH FIELD-ELEMENT SECRETS

We present a PVSS scheme in this section for the conventional key structure in Dlog-based cryptography, i.e., $(sk \in \mathbb{Z}_p, pk = g^{sk} \in \mathbb{G})$. This scheme can be seen as a variant of Fouque-Stern's PVSS [33], while we apply Scrape's technique [19] to improve its verification time and use SoK to embed a creator ID into its transcript.

Building blocks. We briefly recall the building blocks of our PVSS scheme.
Public key encryption. A PKE scheme Σ_{enc} consists of KeyGen, Enc, Dec, satisfying the standard IND-CPA security.
Signature of Knowledge. A signature of knowledge (SoK) scheme SoK for an NP language L consists of three algorithms. $\mathtt{Setup}$ generates a CRS, which is an implicit input of other algorithms. $\mathtt{Sign}(x, w, m)$ 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 [22].
NIZK. We need a NIZK Π_{zk} for showing the well-formedness of the ciphertexts. It consists of $\mathtt{Setup}$, Prove, and Vrfy, for a language L_{pvss} which becomes apparent in the description of construction. We require Π_{zk} to satisfy completeness, simulation soundness, and zero knowledge against PPT adversaries.
The construction.

- $\mathsf{Init}(1^\lambda, n)$. (1) A generator g for the group $\mathbb{G}$ of order p; (2) For every $i \in [n]$, $(ek_i, dk_i) \leftarrow \Sigma_{\mathsf{enc}}.\mathsf{KeyGen}(1^\lambda)$; (3) A setup for SoK; (4) A setup for NIZK Π_{nizk}.
- $\mathsf{Deal}((ek_i)_{i\in[n]}, \mathtt{cid})$. Sample $(a_0, a_1, \ldots, a_t) \leftarrow\$ \mathbb{Z}_p^{t+1}$, define $f(X) = \sum_{i=0}^{t} a_i X^i$, and compute $(A_i = g^{f(i)})_{i\in[0,n]}$, and $(c_i = \Sigma_{\mathsf{enc}}.\mathsf{Enc}(ek_i, f(i)); r_i)_{i\in[n]}$ where r_i is a fresh randomness. Sign $\mathtt{cid}$ with the knowledge of $f(0)$ w.r.t. A_0: $\mathsf{SoK}.\mathsf{Sign}(A_0, f(0), \mathtt{cid}) \to \sigma$. Prove the well-formedness of ciphertexts and obtain a proof $\pi_{\mathsf{zk}} : \Pi_{\mathsf{zk}}$ is for the following language
$$\begin{aligned}\{&\mathtt{statement}\!:\!(A_i, ek_i, c_i)_{i\in[n]}; \mathtt{witness}\!:\!(f(i), r_i)_{i\in[n]} : \\ &A_i = g^{(f(i))} \wedge c_i = \Sigma_{\mathsf{enc}}.\mathsf{Enc}(ek_i, f(i); r_i) \wedge f(i) < p\}\end{aligned} \tag{6}$$
$\mathtt{Trans} = ((A_i)_{i\in[0,n]}, (c_i)_{i\in[n]}, \pi_{\mathsf{zk}}, \{\sigma\}, \{\mathtt{cid}\})$.

- $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \mathtt{Trans})$. It first checks whether

$$\mathsf{SoK.Vrfy}(A_0^{(j)}, \mathtt{cid}^{(j)}, \sigma^{(j)}) = 1, \text{ for all } j \in [m]. \tag{7}$$

  Next, it randomly samples a $(n-t)$-degree polynomial $q(x) \in \mathbb{Z}_p[x]$, and computes the dual code

$$(\mathrm{code}_0^\perp, \ldots, \mathrm{code}_n^\perp), \text{ where } \mathrm{code}_i^\perp = \frac{q(i)}{\prod_{j=0,j\neq i}^{n}(i-j)}. \tag{8}$$

  The dual code is used to check the validity of Scrape's polynomial commitment. See [19]. In our case, it computes $A_0 = \prod_j A_0^{(j)}$ and checks whether

$$\prod_{\tau=0}^{n} A_\tau^{\mathrm{code}_\tau^\perp} = 1. \tag{9}$$

  If the above check passes, it confirms that the exponents of $(A_i)_{i\in[0,n]}$ are from a t-degree polynomial $f(X)$, and $A_i = g_1^{f(i)}$. Then, it checks if π_{zk} is valid proof. It returns 1 if all checks pass; otherwise, it returns 0.
- $\mathsf{getCID}(\mathtt{Trans})$. It returns $\{\mathtt{cid}^{(j)}\}_{j\in[m]}$.
- $\mathsf{PubDriv}(\mathtt{Trans})$. It returns $pk = \prod_j A_0^{(j)}$ and $(pk_i = A_i)_{i\in[n]}$.
- $\mathsf{Dec}(ek_i, dk_i, \mathtt{Trans})$. It returns $sk_i = \mathsf{Paillier.Dec}(ek_i, dk_i, c_i) \bmod p$.
- $\mathsf{Rec}(\{(i, sk_i)\}_{i\in\mathbb{I}})$. It first computes the Lagrange coefficients $\{\lambda_i\}_{i\in\mathbb{I}}$ based on $\mathbb{I}$, and then $sk = \sum \lambda_i sk_i \bmod p$.

A.1 Analysis

Security analysis. The correctness is easy to follow. We establish the other security properties via the following lemmas.

Lemma A.1. *Assuming the soundness of* Π_{zk} *for Eq.6, our PVSS satisfies the soundness.*

Proof. In the soundness definition of PVSS, we require that every transcript passing the public verification can be decrypted to a consistent set of secret shares. According to the public verification algorithm's description, a valid transcript will pass the checking step in Eq.9. This step guarantees, with overwhelming probability, that $A_0, \ldots, A_n$ commit to $f(0), \ldots, f(n)$ for a polynomial f with a degree of up to t, as analyzed in [19]. Subsequently, by the soundness of Π_{zk}, the ciphertexts $c_1, \ldots, c_n$ encrypt to $f(1), \ldots, f(n)$, ensuring that the decrypted values constitute a consistent set of secret shares. Similarly, for a weakly aggregated transcript wTrans □

Lemma A.2. *Assuming the IND-CPA security of the underlying PKE* Σ_{enc}*, the zero-knowledge of* Π_{zk}*, and the security of* SoK*, the PVSS satisfies the strengthened secrecy and the simulation soundness.*

Proof. For clarity, we outline most simulator algorithms in the following.

- $\mathsf{SInit}_{\mathcal{A}}(1^\lambda, n)$. It invokes the simulated setup algorithm of Π_{zk} and SoK to generate $(\mathtt{crs}_{\mathsf{zk}}, \mathtt{tk}_{\mathsf{zk}})$ and $(\mathtt{crs}_{\mathsf{sok}}, \mathtt{tk}_{\mathsf{sok}})$, respectively. Then, the CRS $\mathtt{crs} = (\mathtt{crs}_{\mathsf{zk}}, \mathtt{crs}_{\mathsf{sok}})$ is provided to the adversary $\mathcal{A}$. After receiving the set of corrupted parties C and the encryption keys $\{ek_i\}_{i\in C}$ from $\mathcal{A}$, the simulator generates the encryption/decryption key pairs $(ek_i, dk_i)_{i\in[n]\setminus C}$ for all uncorrupted users. It publishes all encryption keys $(ek_i)_{i\in[n]}$, and sets the trapdoor as $\mathtt{tk} = (\mathtt{tk}_{\mathsf{zk}}, \mathtt{tk}_{\mathsf{sok}})$.
- $\mathsf{SDeal}((ek_i)_{i\in[n]}, pk, \mathtt{tk}, \mathtt{cid})$. First, it sets $A_0 = pk$. For $i \in C$, it samples $y_i \leftarrow\!\$\ \mathbb{Z}_p$, computes $A_i = g^{y_i}$, and encrypts y_i under ek_i: $c_i \leftarrow \Sigma_{\mathsf{enc}}.\mathsf{Enc}(ek_i, y_i)$. For $i \in [n] \setminus C$, it computes A_i via Lagrange interpolation using A_0 and $(A_i)_{i\in C}$, and $c_i \leftarrow \Sigma_{\mathsf{enc}}.\mathsf{Enc}(ek_i, 0)$. Next, it invokes

the simulated signing algorithm of SoK to sign $\texttt{cid}$ and obtains the simulated signature σ. Finally, it invokes the simulated prover algorithm of Π_{zk} to generate the proof π_{zk}, and returns the transcript $\texttt{Trans} = ((A_i)_{i\in[0,n]}, (c_i)_{i\in[n]}, \pi_{\mathsf{zk}}, \{\sigma\}, \{\texttt{cid}\})$.

- $\texttt{SRec}(\texttt{tk}, \texttt{Trans})$. It parses A_0, σ from $\texttt{Trans}$. Then, it runs the extractor algorithm of SoK with the trapdoor key $\texttt{tk}_{\texttt{sok}}$ to extract y^* from σ, such that $A_0 = g^{y^*}$.

We argue for strengthened secrecy through the following hybrid arguments.

Hybrid 0. It runs $\mathsf{Init}_{\mathcal{A}}(1^\lambda, n) \to (\texttt{crs}, C, (ek_i, dk_i)_{i\notin C}, (ek_i)_{i\in C}, st_{\mathcal{A}})$ and $\mathsf{Deal}((ek_i)_{i\in[n]}, \texttt{cid}) \to (\mathsf{Trans}, sk)$. The adversary $\mathcal{A}$ is provided with $(\texttt{crs}, (ek_i)_{i\in[n]}, st_{\mathcal{A}}, \mathsf{Trans})$ as inputs.

Hybrid 1. It is almost identical to Hybrid 0, except that during the initial phase, the setup for Π_{zk} is replaced with the simulated setup algorithm of Π_{zk}, and that the prover algorithm of Π_{zk} in the Deal algorithm is replaced with the simulated prover algorithm.

Hybrid 2. It is almost identical to Hybrid 1, except that during the initial phase, the setup for SoK is replaced with the simulated setup algorithm of SoK, and that the signature of knowledge σ in Trans is generated with the simulated signing algorithm.

Hybrid 3. It is almost identical to Hybrid 2, except that it generates Trans in the following way. (1) Sample $y_i \leftarrow\!\$\ \mathbb{Z}_p$ for all $i \in C$, and sample a t-degree polynomial f, such that $f(i) = y_i$ for $i \in C$. (2) Compute $(A_i = g^{f(i)})_{i\in[0,n]}$, and $(c_i = \Sigma_{\mathsf{enc}}.\mathsf{Enc}(ek_i, f(i)); r_i)_{i\in[n]}$ where r_i is a fresh randomness. (3) Generate the signature of knowledge σ using the simulated signing algorithm. (4) Prove the well-formedness of ciphertexts via the simulated prover algorithm and obtain a proof π_{zk}.

Hybrid 4. It is almost identical to Hybrid 4, except that during generating Trans, the ciphertexts c_i for all $i \in [n] \setminus C$ is generated as $c_i \leftarrow \Sigma_{\mathsf{enc}}.\mathsf{Enc}(ek_i, 0)$.

Hybrid 5. It is almost identical to Hybrid 4, except that in generating Trans, it firstly samples $A_0 \leftarrow\!\$\ \mathbb{G}$ and $y_i \leftarrow\!\$\ \mathbb{Z}_p$ for $i \in C$, interpolates a t-degree polynomial in the exponent based on A_0 and $(A_i = g^{y_i})_{i\in C}$, and obtains A_i for $i \in [n] \setminus C$. After that, the remaining procedures for generating transcript Trans are identical to those in Hybrid 4.

For each hybrid $k \in [0,5]$, we denote the probability that the adversary $\mathcal{A}$ outputs 1 by $\mathsf{P}_{\mathcal{A}}^{(k)}$. Notice that the PVSS scheme satisfies the strengthened secrecy if

$$|\mathsf{P}_{\mathcal{A}}^{(0)} - \mathsf{P}_{\mathcal{A}}^{(5)}| \le \mathsf{negl}(\lambda). \tag{10}$$

It is easy to see that $|\mathsf{P}_{\mathcal{A}}^{(0)} - \mathsf{P}_{\mathcal{A}}^{(1)}| \le \mathsf{negl}(\lambda)$, due to zero-knowledgeness of Π_{zk}. Similarly, we have $|\mathsf{P}_{\mathcal{A}}^{(1)} - \mathsf{P}_{\mathcal{A}}^{(2)}| \le \mathsf{negl}(\lambda)$, from the security of SoK. Also, we have $\mathsf{P}_{\mathcal{A}}^{(2)} = \mathsf{P}_{\mathcal{A}}^{(3)}$ (and $\mathsf{P}_{\mathcal{A}}^{(4)} = \mathsf{P}_{\mathcal{A}}^{(5)}$), as the adversary's views in Hybrid 2 and 3 (resp. Hybrid 4 and 5)are essentially identical. Moreover, it holds that $|\mathsf{P}_{\mathcal{A}}^{(3)} - \mathsf{P}_{\mathcal{A}}^{(4)}| \le \mathsf{negl}(\lambda)$, from the IND-CPA security of the encryption scheme. □

Lemma A.3. *Assuming the security of* SoK*, our PVSS satisfies the simulation soundness.*

Proof. Recall the simulation soundness definition, where we require that the adversary can only issue a challenge transcript that does not contain the CID included in the simulated transcript. Therefore, the signature of knowledge σ in the challenge transcript must be different from the simulated signature of knowledge, and thus, we can extract the witness y^* from the signature. □

Instantiation and performance analysis. The underlying PKE can be instantiated by the Paillier encryption [51], the LWE-based PVW encryption [54], or any other encryption scheme which is accompanied by efficient NIZK Π_{zk} for the language specified in Eq. 6. See [37] for a comprehensive overview on the candidate constructions. Regarding the SoK, we can essentially apply the Schnorr signature while viewing A_0 as the verification key and a_0 as the signing key, respectively. As demonstrated by Gentry et al.[37], the PVW encryption and the associated NIZK enjoy better concrete performance.

Then, assuming the proof size, prover time, and verification time of Π_{zk} are linear to n (which is true of the instantiations discussed above), the transcript size of $\texttt{Trans}$ is $O(n\lambda)$. Both Deal and PubVrfy necessitate $O(n)$ group operations. The computational overhead for the remaining operations is relatively insubstantial, approximating $O(1)$ group operations.

B AN AGGREGATABLE PVSS

We present an aggregatable PVSS scheme in this section, which is a simplified variant of the scheme in [41].

Building blocks. We first introduce a few needed cryptographic building blocks.
Pairing. There is an efficient deterministic algorithm GroupGen which outputs the description of the pairing groups, including $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1)$, where $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ are groups of order p, g_1 is the generator of $\mathbb{G}_1$, $\hat{h}_1$ is the generator of $\mathbb{G}_2$, and $e : \mathbb{G}_1 \times \mathbb{G}_2$ is a bilinear map.

Signature of Knowledge. A signature of knowledge (SoK) scheme SoK for an NP language L consists of three algorithms. Setup generates a CRS, which is an implicit input of other algorithms. $\mathsf{Sign}(x, w, m)$ on inputs a statement x, a witness x, 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 [22].

- $\mathsf{Init}(1^\lambda, n)$. (1) GroupGen $\rightarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1)$; (2) a group element $\hat{u}_1 \leftarrow\$\ \mathbb{G}_2$; (3) the setup for SoK.Setup $\rightarrow \texttt{crs}_{\mathsf{sok}}$. Define $\texttt{crs} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1, \hat{u}_1, \texttt{crs}_{\mathsf{sok}})$. (4) For uncorroputed $i \in [n]$, $dk_i \leftarrow\$\ \mathbb{Z}_p$, and $ek_i = \hat{h}_1^{dk_i}$;
- $\mathsf{Deal}((ek_i)_{i\in[n]}, \texttt{cid})$. Sample $(a_0, a_1, \ldots, a_t) \leftarrow\$\ \mathbb{Z}_p^{t+1}$, define $f(X) = \sum_{i=0}^{t} a_i X^i$, and compute $\hat{u}_2 = \hat{u}_1^{a_0}$, $(A_i = g^{f(i)})_{i\in[0,n]}$, and $(\hat{Y}_i = ek_i^{f(i)})_{i\in[1,n]}$. Sign $\texttt{cid}$ with the knowledge of $f(0)$ w.r.t. A_0: $\mathsf{SoK.Sign}(A_0, f(0), \texttt{cid}) \rightarrow \sigma$. $\texttt{Trans} = ((A_i)_{i\in[0,n]}, (\hat{Y}_i)_{i\in[n]}, \hat{u}_2, \{\sigma\}, \{\texttt{cid}\})$.
- $\mathsf{Agg}(\{\texttt{Trans}_j\}_{j\in[m]}, (ek_i)_{i\in[n]})$. Parse $\texttt{Trans}_j = ((A_i^{(j)})_{i\in[0,n]}, (\hat{Y}_i^{(j)})_{i\in[n]}, \hat{u}_2^{(j)},$ $\sigma^{(j)}, \{\texttt{cid}^{(j)}\})$. Compute $(A_i = \prod_j A_i^{(j)})_{i\in[0,n]}$, $(\hat{Y}_i = \prod_j \hat{Y}_i^{(j)})_{i\in[n]}$, and $\hat{u}_2 = \prod_j \hat{u}_2^{(j)}$. Return $\texttt{Trans} = ((A_i)_{i\in[n]}, (\hat{Y}_i)_{i\in[n]}, \hat{u}_2, \{A_0^{(j)}\}_{j\in[m]} \{\sigma^{(j)}\}_{j\in[m]}, \{\texttt{cid}^{(j)}\}_{j\in[m]})$.
- $\mathsf{PubVrfy}((ek_i)_{i\in[n]}, \texttt{Trans})$. It first checks $\mathsf{SoK.Vrfy}(A_0^{(j)}, \texttt{cid}^{(j)}, \sigma^{(j)})$ for $j \in [m]$. Next, it randomly samples a $(n-t)$-degree polynomial $q(x) \in \mathbb{Z}_p[x]$, and computes the dual code

$$(\mathrm{code}_0^\perp, \ldots, \mathrm{code}_n^\perp), \text{ where } \mathrm{code}_i^\perp = \frac{q(i)}{\prod_{j=0, j\neq i}^{n}(i-j)}.$$

  The dual code is used to check the validity of Scrape's polynomial commitment. See [19]. In our case, it computes $A_0 = \prod_j A_0^{(j)}$ and checks whether

$$\prod_{\tau=0}^{n} A_\tau^{\mathrm{code}_\tau^\perp} = 1. \tag{11}$$

  If the above check passes, it confirms that the exponents of $(A_i)_{i\in[0,n]}$ are from a t-degree polynomial $f(X)$, and $A_i = g_1^{f(i)}$. Then, it checks if $e(A_0, \hat{u}_1) = e(g_1, \hat{u}_2)$, and if $e(g_1, \hat{Y}_i) = e(A_i, ek_i)$ for $i \in [n]$. It returns 1 if all checks pass; otherwise, it returns 0.
- $\mathsf{getCID}(\texttt{Trans})$. It returns $\{\texttt{cid}^{(j)}\}_{j\in[m]}$.
- $\mathsf{PubDriv}(\texttt{Trans})$. It returns $pk = (\prod_j A_0^{(j)}, \hat{u}_2)$ and $(pk_i = A_i)_{i\in[n]}$.
- $\mathsf{Dec}(ek_i, dk_i, \texttt{Trans})$. 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 $sk = \prod sk_i^{\lambda_i}$.

Performance analysis. The transcript size of $\mathtt{Trans}$ is $O((n+m)\lambda)$, where m represents the number of transcripts aggregated into $\mathtt{Trans}$. Both Deal and PubVrfy require $O(n)$ group operations, whereas Agg demands $O(n \log m)$ group operations. The computational costs for other functions are minor, approximately $O(1)$ group operations.

B.1 Security Analysis

The security follows the proofs in [41], which can be reduced to SXDH and BDH assumptions. For clarity, we describe the simulators SInit, SDeal, and SRec in the following, which are implicitly given the proof in [41, Theorem 2].

- $\mathsf{SInit}_{\mathcal{A}}(1^\lambda, n)$. Invoke (1) GroupGen $\rightarrow (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1)$, sample (2) a group element $\hat{u}_1 \leftarrow\$ \mathbb{G}_2$, and run (3) the *simulated* setup for SoK which produces $\mathtt{crs}_{\mathsf{sok}}$ and $\mathtt{tk}_{\mathsf{sok}}$. Define $\mathtt{crs} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, \hat{h}_1, \hat{u}_1, \mathtt{crs}_{\mathsf{sok}})$, provide it to the adversary $\mathcal{A}$, and wait $\mathcal{A}$ to specify the set of corrupted nodes C. (4) for every $i \in [n] \setminus C$, sample $\mu_i \leftarrow\$ \mathbb{Z}_p$, and define $ek_i = \hat{u}_1^{\mu_i}$. Define $\mathtt{tk} = (\mathtt{tk}_{\mathsf{sok}}, \{\mu_i\}_{i\in[n]\setminus C})$.
- $\mathsf{SDeal}((ek_i)_{i\in[n]}, pk, \mathtt{tk}, \mathsf{cid})$. Parse $pk = (g_0, h_0)$. Then, (1) for $i \in C$, sample $a_i \leftarrow\$ \mathbb{Z}_p$, set $A_i = g^{a_i}$, and $\hat{Y}_i = ek_i^{a_i}$. We assume without loss of generality that $|C| = t$. (2) For $i \notin C$, let $A_i = g_0^{\lambda_0(i)} \prod_{j\in C} A_j^{\lambda_j(i)}$, and $\hat{Y}_i = (h_0^{\lambda_0(i)} \prod_{j\in C} \hat{u}_1^{a_j\lambda_j(i)})^{\mu_i}$, where $\lambda_j(i) = \prod_{k\in C, k\neq j} \frac{i-k}{j-k}$. (3) Run the simulation signer algorithm of SoK to generate a simulated signature σ for $\mathtt{cid}$. Output $\mathtt{Trans} = ((A_i)_{i\in[0,n]}, (\hat{Y}_i)_{i\in[n]}, \hat{u}_2 = h_0, \sigma, \{\mathtt{cid}\})$.
- $\mathsf{SRec}(\mathtt{tk}, \mathsf{Trans}) \rightarrow sk$. Parse Trans, and obtain σ. Run the extraction algorithm of SoK with σ and $\mathtt{tk}_{\mathsf{sok}}$, and obtain $s \in \mathbb{Z}_p$ such that $A_0 = g_1^s$. Output $sk = \hat{h}_1^s$.