Scalable and Adaptively Secure Any-Trust Distributed Key Generation and All-hands Checkpointing

Hanwen Feng
School of Computer Science, University of Sydney

Tiancheng Mai
School of Computer Science, University of Sydney

Qiang Tang
School of Computer Science, University of Sydney

Abstract

The classical distributed key generation protocols (DKG) are resurging due to their widespread applications in blockchain. While efforts have been made to improve DKG communication, practical large scale deployments are still yet to come, due to various challenges including broadcast channel scalability and worst-case complaint phase. In this paper, we propose a practical DKG for DL-based cryptosystems, with only (quasi-)linear computation/communication cost per participant, with the help of a public ledger, and beacon; Notably, our DKG only incurs constant-size blockchain storage cost for broadcast, even in the face of worst-case complaints. Moreover, our protocol satisfies adaptive security. The key to our improvements lies in delegating the most costly operations to an *Any-Trust* group. This group is randomly sampled and consists of a small number of individuals. The population only trusts that at least one member in the group is honest, without knowing which one. Additionally, we introduce an extended broadcast channel based on a blockchain and data dispersal network (such as IPFS), enabling reliable broadcasting of arbitrary-size messages at the cost of constant-size blockchain storage, which may be of independent interest.

Our DKG leads to a fully practical instantiation of Filecoin's checkpointing mechanism, in which all validators of a Proof-of-Stake (PoS) blockcahin periodically run DKG and threshold signing to create checkpoints on Bitcoin, thereby enhancing the security of the PoS chain. In comparison with another checkpointing approach of Babylon (Oakland, 2023), ours enjoys a significally smaller monetary cost of Bitcoin transaction fees. For a PoS chain with 2^{12} validators, our cost is merely 0.6% of that incurred by Babylon's approach.

1 Introduction

Better blockchains with threshold cryptography. Threshold cryptography [7, 60] enables a group of nodes to collaboratively perform cryptographic operations, such as message signing or ciphertext decryption, securely even when a minority of nodes are corrupted. In decentralized scenarios like blockchains, it necessitates a distributed key generation (DKG) [34, 52] protocol for setting secret key shares among all parties, eliminating the need of trusted key generation.

Demonstrated by many research work and industry projects, threshold cryptography shows its power to dramatically enhance blockchains in many aspects. Here, we outline a few applications that are for enhancing security aspects, all of which require *all* validators of a blockchain system to execute DKG with a threshold cryptography primitive.

BITCOIN CHECKPOINTING. Long-range attacks [57] are a prominent security challenge in Proof-of-Stake (PoS) blockchain systems. As creating a block in PoS does not consume physical resources, an attacker may effortlessly fork a chain by using the secret keys of previous validators who have left the system, without being punished. Among various options, one main defense mechanism is via checkpointing PoS states into Bitcoin network. In particular, Filecoin project [6, 29] presented an elegant blueprint that allows everyone to distinguish the canonical chain from the attack chain with the help of checkpoints. In their design, a checkpoint is a regular Bitcoin transaction signed by *all* PoS validators through a threshold Schnorr signing protocol [47]. And a DKG is performed for each checkpoint.

CROSS-CHAIN BRIDGE. Cross-chain bridges [48] enable the swapping of assets across different blockchains. A major challenge in existing approaches is how to trustlessly and efficiently verify the status of the other blockchain. Using threshold signatures makes the task easier [16]: *all* validators from one chain can sign the status via a threshold signature, allowing validators from another chain to simply verify the signed messages. One prominent feature in this setting is again DKG needs to be frequently run across the whole validator population, when they are dynamically changing.

CENSORSHIP RESISTANCE AND MEV PROTECTION. Maximal Extractable Value (MEV) attacks [49] are a severe issue for many blockchain systems, where miners could manipulate transaction orders or even exclude valid transactions for their

personal interests, resulting in censorship or even denial of service. Besides obvious reasons in DeFi settings, censorship resistance is also a basic security property of asynchronous consensus protocols [40, 50]. If not properly dealed with, the underlying asynchronous consensus will either loses liveness, or incur large communication blow-up [31]. Threshold encryption provides an elegant solution for MEV/censorship protection. Specifically, all transactions are encrypted under the system's public key and will only be jointly decrypted by all validators after they have been ordered. Many asynchronous consensus took this path [39, 40, 50], and various industry projects have expressed their interests in implementing threshold encryption for MEV protection [55, 61].

DKG: the barrier towards the fantasy. Despite numerous efforts to improving threshold cryptography in the last decade, the promising applications mentioned above remain confined to small-scale prototypes. While many threshold cryptographic primitives like threshold ElGamal and threshold BLS [12] are non-interactive and believed to be scalable, a clear barrier arises from the sophisticated DKG component.

There are a few real-world DKG implementations (*e.g.*, by DFINITY [24], GNOSIS [25], and Drand [27]), which, however, can only scale up to a few dozen nodes [20]. In contrast, for applications mentioned above, it requires dealing with DKG for all validators of the blockchain network (e.g., thousands or more). In the following, we examine the bottlenecks of DKG, identifying the barriers to achieving the fantasy of whole-chain threshold cryptography.

A COMMON DESIGN PARADIGM. For clarity, we include a common paradigm for DKG protocols. In a nutshell, among n participants where up to t could be adversarial, each participant P_i selects 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 collectively engage in a *complaint* phase, where they *broadcast* complaints and identify the set $\mathbb{J} \in [n]$ ensuring that 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 aggregate secret key is $sk = \sum_{j\in\mathbb{J}} sk^{(j)}$.

BURDEN ON BROADCAST. Broadcast channels are essential for DKG protocols to help users reach consensus. Two prominent approaches to a broadcast channel include the use of Byzantine broadcast (BB) protocols [18, 26] or the utilization of pre-existing infrastructure like blockchains. Implementing a large-scale BB protocol can be intricate and susceptible to errors, making the use of established blockchains an attractive, simpler, and modular alternative. This blockchain-based approach naturally aligns with various applications [1–3].

For blockchain-based broadcast channels, we care about blockchain's on-chain storage which is a scarce resource. However, existing DKG schemes incur a large $O(n^2\lambda)$ on-chain storage: traditional schemes [34, 52] employing $O(n\lambda)$-sized polynomial commitments would require quadratic storage during the *delivery* phase already, while schemes with advanced $O(\lambda)$-size polynomial commitments [45, 65] still would cost $O(n^2\lambda)$ storage during the *complaint* phase in the face of $t = \Theta(n)$ Byzantine adversaries. It's worth noting that even though worst-case complaint phases are infrequent occurrences, they must remain feasibly manageable – this is not just a matter of efficiency but also a significant security consideration. Improving the worst-case performance is the major open problem left by previous work [60].

BURDEN ON COMPUTATION. The computation overhead for verifying shares constitutes another bottleneck. Almost all designs necessitate $O(n^2)$ group exponentiation operations. Similar to our discussion about broadcast, traditional schemes require $O(n)$ group exponentiation operations for verifying a single share, resulting in a quadratic computational cost for the *delivery* phase. Advanced schemes still require $O(n^2)$ group exponentiation during the *complaint* phase.

While publicly verifiable secret sharings (PVSS) eliminate the complaint phase, all known schemes already require $O(n)$ group operations for verifying a PVSS transcript, and thus the overall cost is still $O(n^2)$. Their concrete computational overhead is even higher, costing a few hours for a few hundreds of participants [36].

THE CHALLENGE IN THE WEIGHTED SETTING. In DKG and other conventional threshold cryptosystems, all participants are treated equally, and we assume an honest majority of participants. However, in scenarios like PoS blockchains, validators have different *weights*, while a threshold of weights is assumed to belong to honest validators. To bridge this gap, a common approach is to allocate different numbers of subIDs proportional to their weights. Each sub-ID is then treated as an independent participant in threshold cryptosystems. While this approach addresses the security concern, it can lead to an enormous number of sub-IDs. For example, we would need to allocate 674 trillion sub-IDs to 3700 Filecoin validators[1]!

DELEGATING TO A COMMITTEE IS NOT PANACEA. A common strategy to enhance scalability is by selecting a committee and executing the threshold cryptographic systems within this smaller subset. However, this approach is fraught with challenges. An adaptive adversary, once aware of the committee's composition, can compromise the entire group, thereby undermining security. Furthermore, given that each member of the committee is required to contribute multiple times during both key generation and subsequent threshold operations, methods like silent committee sampling (e.g., using a verifiable random function [18]) and assuming memory erasure fail to provide protection against adaptive attackers. Recent advances in the YOSO (You-Only-Speak-Once) MPC realm [11, 35] hint at potential solutions to deter adaptive adversaries targeting the committee. However, these methods come with their own set of challenges, such as a heavy dependence on broadcast channels, necessitating a super hon-

[1] https://filfox.info/en/ranks/power

Table 1: Comparison with the state-of-the-art DKGs.

Schemes	Resilience	Adap.?*	Comm. Cost (total)***			Comp. Cost (per node)**	
			Broad.	P2P/Multi..	Bad†	Good†	Bad†
Pedersen [52]	$1/2$	✓	$O(n\mathcal{B}(n\lambda))$	$O(n^2\lambda)$	-	$O(n^2)$	-
KZG [45, 65]	1/2	✗	$O(n\mathcal{B}(\lambda))$	$O(n^2\lambda)$	$+O(n\mathcal{B}(n\lambda))$	$O(n\log n)$	$+O(n^2)$
GHL [36]‡	$1/2$	✗	$O(n\mathcal{B}(n\lambda)$			$O(n^2)$	
GJM+ [41]‡	$\log n/n$	✗	$O(n\mathcal{B}(\lambda)+\log n\mathcal{B}(n\lambda))$			$O(n\log^2 n)$	
BHK+ [11] §	1/4	✓*	$O(C\mathcal{B}(C\lambda))$	$O(C\mathcal{M}(C^2\lambda))$	-	$O(C^3)$	-
Ours (Sect.3)§	1/2	✓	$O(s\mathcal{B}(n\lambda))$	0	$+O(n\mathcal{M}(s\lambda))$	$O(sn)$	-

*Adap.? asks if the protocol is adaptively secure, and we accept the relaxed definition from [7]; BHK+ achieves the stronger mobile security. **Comp.Cost measures the number of group exponentiation operations performed by each node.
***Comm.Cost measures the communication cost on the different channels, where $\mathcal{B}(\ell)$ (or $\mathcal{M}(\ell)$) denotes the cost of sending ℓ bits via broadcast channel (or multicast channel); We consider $\mathcal{M}(\ell)=O(n\ell)$.
†For both communication and computation, Good (including Broad. and P2P/Multi.) considers the cost without complaints, Bad considers the **extra** cost when facing the maximal number of complaints; "-" represents no asymptotically greater cost.
‡GHL and GJM+ do not have a complaint phase and only use broadcast channels. §BHK+ and ours are committee-based approaches. For ensuring the quality of the committee with high probability (say $1-5\times10^{-9}$, as adopted by Algorand [18]), BHK+ needs the committee size of $C\approx6000$ (estimated based on [10]), while ours only needs $s=20$.

est majority around 75% (which in turn requires a large-size committee with hundreds or thousands of members [10]), or requiring resource-intensive tools like fully homomorphic encryption [37]. While these approaches offer promise, significant gaps remain for our intended applications.

To sum up, even for just a few thousand validators, existing DKG designs demand hundreds of megabytes of on-chain storage coupled with hours of computational effort – a combination that is plainly impractical. Moreover, in the weighted setting, the scale could easily extend to millions, and costs would skyrocket. Recognizing this glaring disparity between practical needs and our available tools, we are compelled to pose the following question:

Is it feasible to devise a DKG protocol scalable to an entire blockchain, secure against adaptive adversaries?

1.1 Our Results

In this work, we have developed a set of techniques that together form an affirmative answer to the question. We outline our results and techniques as follows.

Primary result: A scalable and adaptively secure DKG. Our primary result is a practical adaptively secure DKG protocol for DLog-based cryptographic systems. It features (quasi-)linear per-node computation overhead[2], linear broadcast overhead, even when facing the maximal number of complaints. Moreover, besides the broadcast channel, our DKG only uses a multicast channel (rather than P2P channels), which can be efficiently implemented with gossip protocols for large-scale deployment. We compare our scheme with state-of-the-art efficient DKG constructions in Table 1 and discuss more related works in Section.1.2 [3]. Among them, ours and the YOSO-model DKG (BHK+ [11]) are committee-based approaches that rely on external common randomness. Besides the efficiency differences highlighted in the table, BHK+ generates secret shares for committee members only, meaning that all subsequent threshold cryptographic operations must be done in the YOSO model, which further incurs significant broadcast costs. In contrast, our protocol generates secret shares for the whole population, and conventional threshold cryptography can follow immediately, without consuming additional broadcast bandwidth.

We achieve this result through the following techniques.

SELECTING AN ANYTRUST GROUP AS VSS DEALERS. Our main observation is that it is unnecessary to let all participants act as dealers of VSS. Recall that in the common DKG paradigm, the final secret is $sk=\sum_{j\in\mathsf{Qual}}s^{(j)}$, where $s^{(j)}$ is the secret dealt by a qualified participant $\mathcal{P}_j$. All we need is to ensure the secrecy of sk, and assuming the existence of one honest qualified participant would suffice. Therefore, we propose utilizing a group of participants, known as an "any-trust" group (as introduced in the context of anonymous communication [63]), where we can trust at least one honest member as VSS dealers. Since a group member does not need to keep any secret information beyond its private keys, we can employ techniques from [18, 21], including verifiable random function-based sortition [18] and forward-secure

[2] Evaluating $O(n)$-degree polynomials at $O(n)$ points inherently causes $O(n\log n)$ computation. However, we only need $O(n)$ expensive group exponentiation operations.

[3] Asynchronous DKGs [4, 20, 32] are not included in the table, as their implementations cannot simply leverage a broadcast channel. Instead, they rely on $O(n)$ instances of reliable broadcast protocols [13] and incur $O(n^3\lambda)$ communication cost.

signatures [21], to defend against *adaptive corruption*. It is also important to note that the size s of an any-trust group can be as small as a few tens, which is in stark contrast to that of a group with an honest majority, which can be up to hundreds or thousands. Looking ahead, we commit to a polynomial by committing to its n evaluations (as Scrape [15] does). Then, with an any-trust group as dealers, the broadcast cost is $s\mathcal{B}(n\lambda)$, and the per-node computation is $O(sn)$, *before* the complaint phase.

OPTIMIZED COMPLAINTS FOR PUBLIC VERIFIABILITY. Using an any-trust group as dealers necessitates a fundamental shift in the complaint phase, as the dealers are not supposed to respond to any complaints. To work around this issue, we adopt publicly verifiable and unforgeable complaints. Particularly, we let the dealers broadcast all encrypted shares, which ensures the availability of ciphertexts. This step inherently requires the broadcast cost of $s \cdot \mathcal{B}(n\ell)$, and using a constant-size polynomial commitment cannot help. Then, if a receiver obtains an incorrectly decrypted share, it can complain against the dealer by disseminating a NIZK proof of decryption along with the share. By doing this, everyone can disqualify a dealer by verifying a complaint without further interaction with the dealer. Moreover, the publicly verifiability and unforgeability of complaints enable a more efficient publishing method. Particularly, after each node disseminates the complaints using a multicast channel, we sample an anytrust group again and let the group members deduplicate the complaints and broadcast them, which guarantees that all malicious dealers will be disqualified. With the optimized complaint phase, the broadcast and computation cost of our DKG will not blow up even if there are many complaints. More details are discussed in Sect. 3.

Add-on 1: A practical extended broadcast channel. The broadcast cost of our DKG is $s \cdot \mathcal{B}(n\lambda)$. When implementing the broadcast channel with a blockchain, we need to write $O(sn\lambda)$ bits into the blockchain. While the cost may be acceptable when n is a few thousand, it is certainly a bottleneck for extremely large n. Therefore, we present a practical extension to the blockchain-based broadcast channel, by leveraging a multicast channel and a data dispersal network (DNN) like IPFS [62]. Our design is simple and modular, retaining the major benefits of using blockchain. Let $\mathcal{PB}(m)$ be the cost of posting m bits to the blockchain and $\mathcal{R}(s)$ be the cost of registering s data blocks to the DNN. Then, the cost of s senders each sending ℓ bits via our broadcast channel is

$$s \cdot \mathcal{B}(\ell) = s\mathcal{PB}(\lambda) + O(sn\ell) + c\mathcal{PB}(\lambda + s) + n\mathcal{R}(s), \quad (1)$$

where c is the size of an honest majority committee sampled from the population. We can choose $c = 296$, which guarantees the honest majority with high probability $(1 - 5 \times 10^{-9})$, when $2/3$ nodes/weights of the whole population are honest[4]. Note that the blockchain storage cost is independent of the message length ℓ, so a sender can broadcast an arbitrarily long message while incurring constant on-chain storage cost.

Our design is inspired by Byzantine Broadcast extensions [51], which invoke a few instances of Byzantine Broadcast/Agreement for λ bits plus some P2P communications to realize Byzantine Broadcast for a large amount of data. In our design, we essentially view the blockchain as an oracle to reach a consensus about the status of data while using DNN to guarantee availability. Though it may be folklore to write digests alone into a blockchain to save bandwidth, we are not aware of any design with a formal guarantee of agreement. We believe this component may be of independent interest. More details are in Sect.4.

Add-on 2: Optimized sub-ID allocation for the weighted setting. While the traditional sub-ID allocation method precisely preserves the portion of each party's power, we have noticed and leveraged a gap between the usual assumption on the honest participant's weight ratio (assumed to be more than 2/3 due to other components of the system) and the honest ratio needed in threshold cryptography (usually just above 1/2). Therefore, we propose a lossy-yet-qualified allocation, guaranteeing that more than half of the sub-IDs will be issued to honest participants if they possess over 2/3 of the weights. With this simpler goal in mind, we have devised a more efficient sub-ID allocation method, requiring only 1688 sub-IDs instead of 674 trillion for the 3700 Filecoin validators. In comparison with concurrent work in [23], ours issues fewer sub-IDs for large validator sets like Filecoin's. More details can be found in Sect.5.

Application: All-hands checkpointing. We then apply our techniques to realize the checkpointing mechanism of Filecoin [6] that require all validators to participate. After our optimized sub-ID allocation, we need to execute a DKG and a threshold Schnorr signature [56] among these 1688 sub-IDs to create a checkpoint. With our Any-Trust DKG, the DKG phase only incurs around 1.6MB of broadcast messages in total. Each node can complete all computations in just a few seconds, even when facing the maximum number of complaints. Regarding the threshold signature, we use Any-Trust DKG again to generate the nonce in the GJKR [34] signing protocol, resulting in a non-interactive threshold signing protocol (after nonce-generation), eliminating the potential single point of failures in coordinator-based protocols like FROST [47].

Compared with a recent checkpointing scheme, Babylon [58], which uses an aggregatable signature instead of a threshold signature, ours/Pikachu only requires exactly *one* Bitcoin transaction for each checkpoint. In their scheme, the number of Bitcoin transactions per checkpoint grows linearly. This difference reflects on monetary cost. As an example with Filecoin, the estimated Bitcoin transaction fee per annum would be 2152.3 USD for ours and 344,373.1 USD for theirs, especially when we choose to address the potential single point of failure in Babylon. More details can be found in Sect.6.

[4]The parameter is calculated with the program in [22]

Implementation and Evaluation. We implemented our protocol in Java and deployed it on AWS EC2 instances with 32, 64, and 128 nodes. The results demonstrate that our protocol scales effectively and completes within a few seconds (excluding ledger waiting time which could vary depending on the blockchain). Additionally, we conducted computational time tests for various values of n, ranging from 2^8 to 2^{15}. In comparison with the state-of-the-art DKG protocol KZG [45], our protocol's performance in both the good-case and worst-case scenarios is comparable to or even superior to KZG's performance in the good-case scenario. Notably, KZG's cost in the worst-case scenario experiences a significant increase.

1.2 Related Works

Distributed Key Generation (DKG) has been a prominent area of research for several decades. Pedersen's seminal work [52] established the foundation in this field by introducing an efficient protocol for Dlog-based cryptosystems. This protocol builds upon Feldman's Verifiable Secret Sharing (VSS) [28]. Within this scheme, each participant collaboratively runs n instances of Feldman's VSS, taking on the role of the dealer in one of these instances.

In the VSS framework established by Feldman, the dealer is required to broadcast a commitment to a polynomial, while distributing the shares privately among all participants. Given that the commitment's size is proportional to $O(n\lambda)$, the resultant communication overhead becomes $O(n\mathcal{B}(n\lambda))$. Additionally, Pedersen's DKG involves a complaint phase where participants broadcast any grievances against dishonest dealers. If a participant were to lodge multiple complaints concurrently, the communication overhead of this phase is likewise $O(n\mathcal{B}(n\lambda))$. It is vital to highlight that during this phase, each participant may validate up to $O(n^2)$ shares. In Feldman's VSS, the computational effort to validate a single share is equivalent to $O(n)$ group operations. This implies a per-node computational burden before the complaint phase of $O(n^2)$, which can potentially amplify to $O(n^3)$ during the complaint process.

A majority of DKG architectures conform to the joint-VSS model. In essence, any innovative VSS protocol can be adapted into a new DKG protocol. Furthermore, given that VSS can be constructed using polynomial commitments, any polynomial commitment scheme can be evolved into both a VSS and consequently a DKG. A significant advancement in this field was made by Kate et al. [45] who proposed the first polynomial commitment (abbreviated as KZG) with a commitment size of $O(\lambda)$. This innovation ensures that prior to the complaint phase, the communication overhead can be reduced to $O(n\mathcal{B}(\lambda))$. A notable feature of the KZG polynomial commitment is its efficiency in verifying shares; the computational cost for verifying a single share is a mere $O(1)$. This denotes that the computational overhead for each node, in terms of verification before the complaint phase, is simply $O(n)$ in group operations, but this can rise to $O(n^2)$ during the complaint process. Historically, the computational load for producing a polynomial commitment was believed to be $O(n^2)$ [60]. However, a recent exploration by Zhang et al. [65] revealed that the computational overhead for generating a KZG commitment can be streamlined to $O(n\log n)$. It's noteworthy that although KZG requires a CRS setup, there have been other efforts [64, 65] that prioritize efficient polynomial commitments without relying on a trusted setup, but these don't match KZG's efficiency.

Fouque and Stern [30] offered a solution that sidestepped the necessity for a complaint phase by incorporating publicly verifiable secret sharing (PVSS). In the event that a PVSS transcript consists of $O(n)$ ciphertexts, the communication overhead will naturally be $O(n\mathsf{BB}_n(n\lambda))$ should every participant choose to broadcast this transcript. Historically, the validation of a PVSS transcript required an overhead of $O(n^2)$, suggesting that the per-node computational overhead in DKG might ascend to $O(n^3)$. However, this obstacle was surmounted by Cascudo and David with their Scrape protocol [15], which introduced a PVSS methodology that caps the verification duration at $O(n)$. It's worth highlighting that Scrape's strategy is versatile, and can be harnessed to improve numerous previous methodologies, including that of Pedersen's, ensuring that computational overhead during the complaint phase is kept at $O(n^2)$ and doesn't spike to $O(n^3)$. A distinctive branch of research, as evident in works like [36], has pivoted towards enhancing the tangible performance of PVSS.

Gurkan et al. [41] leveraged an aggregatable PVSS combined with gossip protocols to craft a publicly verifiable DKG. Their communication overhead is streamlined to $n\mathcal{B}(\lambda) + \log n \cdot \mathcal{B}(n\lambda)$ as opposed to $n\mathcal{B}(n\lambda)$, with their per-node communication overhead being $O(n\log^2 n)$. It's pertinent to note, however, that their model can only accommodate $O(\log n)$ Byzantine nodes. Another noteworthy contribution by Benhamouda et al. [11] delves into DKG within the YOSO model, where anonymous committees are periodically selected to oversee each protocol round. Yet, existing techniques for choosing the anonymous committee either mandate a super-majority of honest nodes (approximately 75% of them) [10] or hinge on advanced tools such as fully homomorphic encryption [37]. Furthermore, as successive committees remain anonymous, inter-committee communication is heavily dependent on a broadcast channel.

Beyond endeavors aimed at bolstering the efficiency of DKG, various research initiatives have tackled this challenge from different perspectives. Gennaro et al. [34] pinpointed vulnerabilities in Pedersen's DKG where the secret key distribution could be manipulated by adversaries. They addressed this flaw by achieving complete secrecy, albeit with a higher computational overhead. Gurkan et al. [41] conceptualized a milder form of secrecy, coined as "key-expressability", which assumes that adversaries can influence key distribution but

within predetermined constraints. They postulated that a key-expressable DKG suffices for many applications, with multiple DKG architectures, including Pedersen's [52], Fouque-Stern's [30], and our own, fitting this criteria. Another remarkable contribution by Canetti et al. [14] introduced a DKG protocol with adaptive security, a departure from our model and numerous others that ensure security only against static adversaries. Bacho and Loss's recent work [7] put forth an oracle-aided adaptive definition and ascertained that several protocols, including [30, 52], conform to this definition in the algebraic group model. Our model also complies with this adaptive security definition.

Lastly, some novel research efforts [4, 20, 32] have pivoted towards DKG within asynchronous networks. These designs adopt the joint-VSS blueprint and depend on an asynchronous broadcast protocol, referred to as "reliable broadcast" [13], to guarantee verifiability, yet they encounter the cubic computational challenge. Notably, Das et al. [20] showcased the inaugural asynchronous DKG with a communication overhead of $O(n^3\lambda)$ for field-element secrets, whereas Abraham et al. [4] furnished an adaptively secure asynchronous DKG with identical complexity.

2 Model and Preliminaries

Communication model. We assume the network is synchronous and protocols proceed by rounds. Every participant has access to a multicast channel and a broadcast channel which have different guanranteed delivery time. They both achieve *validity*, while broadcast channel additionally guarantees *agreement*. We always let the sender also be a receiver.
VALIDITY. When an honest node send a message via this channel, all other honest nodes can receive this message by the end of the multicast round (or broadcast round).
AGREEMENT. At the end of a broadcast round, honest receivers always receive the same message from this channel, even when the sender is Byzantine.

Adversarial model. Prior to protocol execution, every node honestly generates their public key/secret key pairs and sends public keys to all other nodes. After the setup, the adversary can adaptively corrupt any node during the protocol execution and control their subsequent behaviours. Particularly, the adversary controls what messages a corrupted node will send in the same round it gets corrupted. However, messages that were already multicasted or broadcasted by node i before i become corroputed *cannot be retracted*.

Notations and assumptions. 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. A function $f(n)$ is deemed negligible in n, denoted by $f(n) \leq \text{negl}(n)$, if for every positive integer c, there exists an n_0 such that for all $n > n_0$, $f(n) < n^{-c}$. For a set $\mathbb{X}$, the notation $x \leftarrow\!\$\, \mathbb{X}$ signifies sampling x uniformly from $\mathbb{X}$. $A(x_1, x_2, \cdots; r)$ 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 $y = A(x_1, x_2, \cdots; r)$. Adversaries are assumed to be probabilistic polynomial time (PPT).

Distributed Key Generation (DKG): An (n,t)-DKG for DLog-based cryptography is an interactive protocol involving n parties. At the end of an execution, all honest parties possess a common public key $pk \in \mathbb{G}$ and a list of public key shares $(pk_1, \ldots, pk_n)$, while each of them holds a secret share $sk_i \in \mathbb{Z}_p$. We follow Bacho and Loss's DKG definition [7], dubbed *oracle-aided algebraic security*. This definition captures active attackers who may adaptively corrupt parties during the protocol execution, and a DKG satisfying this definition would suffice for instantiating the key generation part of threshold BLS [7]. This definition considers attackers that can be modeled as algebraic algorithms.

Definition 1 (Algebraic Algorithm)**.** An algorithm A is called algebraic over a group $\mathbb{G}$ if all group element $\zeta \in \mathbb{G}$ that A outputs, it additionally outputs a vector $\vec{z} = \{z_0, \ldots, z_m\}$ of integers in $\mathbb{Z}_p$ such that $\zeta = \prod_i g_i^{z_i}$, where $(g_1, \ldots, g_m)$ is the list of group elements that A has received so far.

Definition 2. Let Π be a protocol among n parties $P_1, P_2, \ldots, P_n$ where P_i outputs a secret key share sk_i, a vector of public key shares $(pk_1, \ldots, pk_n)$, and a public key pk. Π is a secure DKG for a DL cryptosystem over a group $\mathbb{G}$ of a prime order p, if it satisfies the following properties.

- **Consistency:** Π is t-consistent, if despite that at most t parties have been corrupted, the honest parties can output the same public key pk and the same vector of public key shares $(pk_1, \ldots, pk_n)$.
- **Correctness:** Π is t-correct, if despite that at most t parties have been corrupted, there is a t-degree polynomial $f(x) \in \mathbb{Z}_p[X]$, such that for every $i \in [n]$, $pk_i = g^{f(i)}$, every honest $\mathcal{P}_i$ has $sk_i = f(i)$, and the public key is $pk = g^{f(0)}$.
- **Oracle-aided Algebraic Simulatability:** Π has $(t, k, T_{\mathcal{A}}, T_{\mathsf{sim}})$-oracle-aided algebraic simulatability if for every adversary $\mathcal{A}$ that runs in time at most $T_{\mathcal{A}}$ and corrupts at most t parties, there exists an algebraic simulator Sim that runs in time at most T_{sim}, makes $k-1$ queries to oracle $\mathsf{DL}_g(\cdot)$ and satisfies the following properties:

  On input $\zeta = (g^{z_1}, \ldots, g^{z_k})$, Sim simulates the role of the honest participants in an execution of Π. Upon an honest party $\mathcal{P}_i$ being corrupted, the simulator needs to return the internal state of $\mathcal{P}_i$ to the adversary.

  On input $\zeta = (g^{z_1}, \ldots, g^{z_k})$, let g_i denote the i-th query by Sim to $\mathsf{DL}_g(\cdot)$. Let $(\hat{a}_i, a_{i,1}, \ldots, a_{i,k})$ be the

corresponding algebraic coefficients of g_i, *i.e.*, $g_i = g^{\hat{a}_i}\prod_{j=1}^{k}(g^{z_j})^{a_{i,j}}$ and set $(\hat{a}, a_{0,1}, \ldots, a_{0,k})$ as the algebraic coefficients of pk. Then, the following matrix over $\mathbb{Z}_p$ is invertible

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

Whenever Sim completes a simulation of an execution of Π, we call L the simulatability matrix of Sim.

Denote by $\mathsf{view}_{\mathcal{A},y,\Pi}$ the view of $\mathcal{A}$ in an execution of Π conditioned on all honest parties outputting $pk = y$. Denote by $\mathsf{view}_{\mathcal{A},\zeta,y,\mathsf{Sim}}$ the view of $\mathcal{A}$ when interacting with Sim on input ζ, conditioned on Sim outputting $pk = y$. Then, for all y and all ζ, $\mathsf{view}_{\mathcal{A},y,\Pi}$ and $\mathsf{view}_{\mathcal{A},y,\Pi}$ are computationally indistinguishable.

Note that the adversary $\mathcal{A}$ does not have to be fully algebraic. Instead, being algebraic related to pk and queries $\mathsf{DL}_g(\cdot)$ would suffice, as discussed in [7].

Additionally, we consider 'key-expressibility,' as introduced in [41], against static attackers. This property is suitable for instantiating the key generation of BLS, ElGamal, and Schnorr [41, 56].

Definition 3 (Key-expressability [41])**.** A DKG protocol is key-expressable, if for every static PPT adversary $\mathcal{A}$ that corrupts up to t nodes, there exists a PPT simulator Sim, such that on input of a uniformly random element $pk' \in \mathbb{G}$, produces $\alpha\mathbb{Z}_p$, $sk_1 \in \mathbb{Z}_p$, $pk_1 = g^{sk_1} \in \mathbb{G}$, and a view which is indistinguishable from $\mathcal{A}$'s view from a run of the DKG protocol that ends with $pk = pk'^{\alpha} \cdot pk_1$.

Verifiable random function based sortition. A verifiable random function (VRF) is a pseudorandom function whose outputs can be publicly verified using the evaluator's public key. Throughout this paper, we use VRFs for the sole purpose of cryptographic sortition, and thus we present VRF-based sortition below.

- $\mathsf{Setup}(1^\lambda)$. Each user generates their VRF key pair (vk, sk) and publishes vk. A public randomness rand is sampled independent of the key generation.
- $\mathsf{Sortition}(vk, sk, \mathsf{rand}, \mathsf{event}, \mathsf{ratio})$. A user with (vk, sk) evaluates the VRF on the input of $(\mathsf{rand}\|\mathsf{event})$ and obtains y and a proof π. It checks if $\frac{y}{\mathsf{max}} \leq \mathsf{ratio}$, where max is the max value in the range of the VRF. If failed, abort. Otherwise, return (y, π) as the credential of being selected.
- $\mathsf{Vrfy}(vk, \mathsf{rand}, \mathsf{ratio}, \mathsf{event}, \mathsf{credential})$. It verifies the credential by validating the VRF output y and checking if $\frac{y}{\mathsf{max}} \leq \mathsf{ratio}$.

In the above description, ratio denotes the ratio of the expected committee size to the whole group size, and the expected committee size is determined by the expected ratio of honest nodes to the committee.

Throught this paper, we take the DDH-based VRF scheme from [38] as our instantiation, whose evalutation proof only consists of 2 elements in $\mathbb{Z}_p$ and one group element. Its evaluation algorithm costs 1 group exponenet operation, while its verification algorithm requries 4 group exponentiation operations.

Verifiable multi-recipient encryption. We use the *hybrid version* of ElGamal encryption as the PKE. Let g be a generator of $\mathbb{G}$, and let Hash as a hash function modeled as a random oracle. The key generation algorithm Gen outputs $(ek = g^x, dk = x)$, where $x \leftarrow\$ \mathbb{Z}_p$. The encryption algorithm $\mathsf{Enc}(ek, m)$ first samples $r \leftarrow\$ \mathbb{Z}_p$, and computes the ciphertext $(c_0, c_1) = (g^r, \mathsf{Hash}(ek^r) \oplus m)$. The decryption algorithm $\mathsf{Dec}((c_0, c_1), dk)$ returns $m = \mathsf{Hash}(c_0^{dk}) \oplus c_1$.

ElGamal encryption can be extended to an efficient *multi-recipient encryption* [9]. Namely, when someone wants to encrypt a sequence of messages $(m_1, m_2, \ldots, m_n)$ under a sequence of public keys $(ek_1, ek_2, \ldots, ek_n)$, it can reuse the randomness, *i.e.*, it computes the ciphertext

$$\mathsf{MREnc}(ek_1, \ldots, ek_n, m_1, \ldots, m_n) := (c_0 = g^r, c_1, \ldots, c_n), \tag{2}$$

where $\forall i \in [n], c_i = \mathsf{Hash}(ek_i^r) \oplus m_i$. In this way, the ciphertext size is the plaintext size plus the size of a group element, greatly saving communication costs. We remark assuming a random oracle Hash can greatly simplify the security proof in the presence of adaptive receiver corruption, as observed in [42, 43]; it will become apparent in our security analysis.

There is a proof system for decryption correctness $\{\mathsf{Prove}, \mathsf{Vrfy}\}$. With $\mathsf{Prove}(c_0, c_i, dk_i, m) \to \Gamma$, one can prove that m is the correct decryption from (c_0, c_i), by publishing a proof $\Gamma = (m, c_0^{dk_i}, \pi)$. Here π demonstrates the discrete logarithm of $c_0^{dk_i}$ w.r.t c_0 is equal to that of ek_i w.r.t. g, which is commonly known as DLEQ proof (equality of discrete logarithms) [17]. $\mathsf{Vrfy}(c_0, c_i, \Gamma)$ checks if m is the correct decryption from (c_0, c_1) w.r.t. ek_i. π consists of 2 elements in $\mathbb{Z}_p$, and verifying it takes 4 group exponentiation operations.

Forward-secure digital signature. A forward-secure signature scheme $\mathsf{FS}.\Sigma$ consists of four algorithms: (1) $\mathsf{Gen}(1^\lambda) \to (\mathsf{FS}.vk, \mathsf{FS}.sk[1])$ generates a verification key and the initial signing key; (2) $\mathsf{Update}(\mathsf{FS}.sk[i]) \to \mathsf{FS}.sk[i+1]$ updates the signing key at round i to the signing key at round $i+1$; (3) $\mathsf{Sign}(\mathsf{FS}.sk[i], m) \to \sigma$ generates a signature σ for the message m; (4) $\mathsf{Vrfy}(\mathsf{FS}.vk, i, \sigma, m) \to b$ determines if σ is a valid signature for m created by the signing key at round i. A forward-secure signature scheme guarantees the unforgeability of signatures at rounds $i < i^*$, even when the adversary has access to signing oracles at any round and corrupts the signing key at the i^*-th round.

3 Our DKG Protocol

We present our Any-Trust DKG protocol in this section.

High-Level Overview. At the heart of our approach lies a fundamental shift in the joint-VSS based DKG (JV-DKG) strategy. We identify that in JV-DKG, ensuring *secrecy* only requires a single honest VSS dealer, while previous schemes let all nodes deal secrets. With this insight, we choose a compact group (called the "any-trust group") which, with high probability, contains at least one honest node, and assign only these group members as dealers. While an adaptive adversary has the resources to compromise an entire committee, we leverage techniques from [18, 21] to address the challenge. Specifically, we employ VRF-based sortition to anonymously/secretly select the group, preventing adversaries from identifying members before their message broadcasts. For security, group members erase their internal secrets used to generate VSS transcripts, ensuring adversaries, even after compromising a member, cannot access its contributed secret. Additionally, we employ a forward-secure signature [44], compelling honest entities to delete their old signing keys. This ensures that post-corruption, adversaries can't send additional messages in the same round, preserving the contributions of honest dealers to the final secret key – a critical factor for maintaining secrecy.

The complaint phase presents its own set of technical challenges, especially because (1) dealers are expected to delete their secret states, inhibiting their ability to address complaints, and (2) we aim to disqualify all dishonest dealers while minimizing the communication burden on the broadcast channel. To address the first issue, we introduce "verifiable complaints", enabling all to disqualify a dealer without necessitating further communication. Notably, dealers broadcast all encrypted shares, ensuring a unified view of these shares for all nodes. Each complaint includes a decryption proof, facilitating verification by cross-referencing the broadcasted ciphertext for discrepancies in decrypted shares. To handle the second issue, we use an additional anytrust group to collate all complaints from the multicast channel, broadcasting only unique complaints. This structure ensures that only $O(s^2)$ complaints (where s is the size of the anytrust group) are transmitted via the broadcast channel, yet all dishonest dealers are reported.

While PVSS schemes might offer a way to bypass the complaint phase, we avoid them due to potential computational overhead. Furthermore, there exists no adaptively secure PVSS for a secret key in $\mathbb{Z}_p$. In terms of polynomial commitments, we adopt techniques from [15], directly committing to polynomial coefficients, which streamlines the verification process for multiple shares under one commitment. We highlight that advanced polynomial commitments [45] don't significantly reduce our communication costs, as we consistently place all encrypted shares in the broadcast channel to facilitate verifiable complaints.

Setup. Given the security parameter λ, the number of participants n, and the corruption bound t (where the adversary can corrupt up to t parties), configure the system as follows:

GROUP DESCRIPTION: Based on the security parameter λ, define the group order as a prime p. Describe the group $\mathbb{G}$ of order p and its generator g. The resulting public key will belong to group $\mathbb{G}$ and will have the form g^{sk}.

PKI SETUP: Every participant $\mathcal{P}_i$ produces three key pairs: (ek_i, dk_i) for PKE, (rvk_i, rsk_i) for VRF, and $(\mathsf{FS}.vk_i, \mathsf{FS}.sk_i[1])$ for the forward-secure digital signature scheme.

RANDOM COIN: Uniformly select a string $\mathsf{rand} \leftarrow\!\!\$\ \{0,1\}^\lambda$ that is independent of all users' public keys.

Determine the value of ratio for VRF-based sortition. Given n participants executing the sortition algorithm with ratio, the chosen committee will form an any-trust group. We assume the required configurations for the foundational channels are established during this setup phase.

Protocol Details. Post-setup, all participants collaboratively run our DKG protocol, detailed in Fig.1, utilizing building blocks such as the PKE scheme PKE, the forward-secure signature FS, and the VRF-based sortition VRF. All these components are outlined in Sect.2.

The protocol initiates with a broadcast round, transitions to a multicast round, and concludes with a final broadcast. A brief overview of each round is:

- **Round 1.** Nodes initially determine if they're selected as dealers. If not, they refresh the secret key and exit the round (lines 1-3). Chosen dealers sample a t-degree polynomial f to decide secret shares $sk_i = f(i)$, commit to $sk_0, \ldots, sk_n$, and encrypt shares $sk_1, \ldots, sk_n$ using others' encryption keys (lines 5-7). Dealers then sign the commitments and ciphertexts, update their signing keys, erase secret information, and broadcast the signed commitments and ciphertexts (lines 8-12).

- **Round 2.** Nodes receive the broadcasted messages (line 1). For every message, they authenticate the signature (line 6), if failed, move to the next message. Otherwise, they validate its format, and the VRF sortition certificate (lines 7-8). Moreover, they ascertain if the committed values match valid coefficients of a t-degree polynomial, utilizing methods from [15] (lines 3-4 and 9-10). If a transcript fails verification, the dealer is instantly disqualified (line 11). Otherwise, they check the decrypted share's validity against the commitments and, if inconsistent, generate a verifiable complaint against the dealer (lines 12-15). All complaints are multicast.

- **Round 3.** Nodes first verify if they are selected as senders (lines 1-4). If so, they collect and verify all complaints (using ciphertexts received from line 1 of round 2), de-duplicate them, and curate a complaint list documenting all complained dealers (lines 5-13). They then sign and broadcast this complaint list (lines 15-17).

Round 1 (broadcast): each P_i **do**:

1: $\mathsf{VRF.Sortition}(rvk_i, rsk_i, \mathsf{rand}, \text{“deal”}, \mathsf{ratio}) \to \mathsf{CR}_i^{\mathsf{deal}}$
2: **if** $\mathsf{CR}_i^{\mathsf{deal}} = \perp$,
3: **then** $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$, **exit Round 1**
4: // only elected users continue the followings.
5: sample $(a_0, a_1, \ldots, a_t) \leftarrow\$ \mathbb{Z}_p^{t+1}$, define $f(X) = \sum_{\tau=0}^{t} a_\tau X^\tau$
6: commit $(\mathsf{cm}_j = g^{f(j)})_{j\in[0,n]}$ // commit to polynomial $f(X)$
7: encrypt $\mathsf{PKE.MREnc}((ek_i)_{i\in[n]}, (f(i))_{i\in[n]}) \to (c_0, \ldots, c_n)$
8: denote $\mathsf{trans}_i \leftarrow (\mathsf{CR}_i^{\mathsf{deal}}, (\mathsf{cm}_j)_{j\in[n]}, (c_j)_{j\in[n]})$
9: sign $\mathsf{FS.Sign}(\mathsf{FS}.sk_i[1], \mathsf{trans}_i) \to \sigma_i$
10: $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$
11: **erase** $\mathsf{FS}.sk_i[1], f(X), (f(i))_{i\in[0,n]}$, and encryption randomness
12: **broadcast** $(i, \mathsf{trans}_i[1], \sigma_i[1])$

Round 2 (multicast): each P_i **do**:

1: **receive** $\{(j, \mathsf{trans}_j[1], \sigma_j[1])\}_{j\in\mathbb{D}}$, for $\mathbb{D} \subset [n]$
2: set $\mathbb{D}_1, \mathbb{D}_2, \mathbb{D}_3, \mathbb{C} = \emptyset$
3: sample an $(n-t)$-degree polynomial $q(X) \in \mathbb{Z}_p[x]$, compute
4: $\mathsf{cm}_\tau^\perp = \dfrac{q(\tau)}{\prod_{j=0, j\neq\tau}^{n}(\tau - j)}, \forall \tau \in [0,n]$ // the dual code [15]
5: **for** $j \in \mathbb{D}$
6: **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 1, \sigma_j[1], \mathsf{trans}_j[1]) = 0$, **then continue**
7: **if** parse $\mathsf{trans}_j[1] = (\mathsf{CR}_j^{\mathsf{deal}}, (\mathsf{cm}_\tau^{(j)})_{\tau\in[n]}, (c_\tau^{(j)})_{\tau\in[n]})$ failed
8: $\vee\, \mathsf{VRF.Vrfy}(rvk_j, \mathsf{rand}, \mathsf{ratio}, \text{“deal”}, \mathsf{CR}_j^{\mathsf{deal}}) = 0$
9: //check if $(\mathsf{cm}_\tau^{(j)})_{\tau\in[n]}$ commits to a t-degree polynomial
10: $\vee \prod_{\tau=0}^{n} (\mathsf{cm}_\tau^{(j)})^{\mathsf{cm}_\tau^\perp} \neq \mathbf{1}_{\mathbb{G}}$ // the identity element of $\mathbb{G}$
11: **then** $\mathbb{D}_1 = \mathbb{D}_1 \cup \{j\}$ // disqualify j immediately
12: **elseif** $\mathsf{PKE.Dec}(dk_i, c_i^{(j)}) = sk_i^{(j)} \wedge g^{sk_i^{(j)}} \neq \mathsf{cm}_i^{(j)}$
13: //generate a complaint, and update the complaint list
14: **then** $\mathsf{PKE.Prove}(c_0^{(j)}, c_i^{(j)}, dk_i, sk_i^{(j)}) \to \Gamma_j$,
15: $\mathbb{D}_2 = \mathbb{D}_2 \cup \{(j, \Gamma_j)\}$
16: //otherwise, update the candidate output list
17: **else** $\mathbb{D}_3 = \mathbb{D}_3 \cup \{j\}, \mathbb{C} = \mathbb{C} \cup \{(j, ((\mathsf{cm}_\tau^{(j)})_{\tau\in[0,n]}, sk_i^{(j)}))\}$
18: $\mathbb{D}_2 \to \mathsf{trans}_i[2], \mathsf{FS.Sign}(\mathsf{FS}.sk_i[1], \mathsf{trans}_i[2]) \to \sigma_i[2]$
19: **if** $\mathbb{D}_2 \neq \emptyset$, **then multicast** $(i, \mathsf{trans}_i[2], \sigma_i[2])$

Round 3 (broadcast): each P_i **do** :

1: **receive** $\{(j, \mathsf{trans}_j[2], \sigma_j[2])\}_{j\in\mathbb{R}_1}$, for $\mathbb{R}_1 \subset [n]$
2: $\mathsf{VRF.Sortition}(rvk_i, rsk_i, \mathsf{rand}, \text{“agree”}, \mathsf{ratio}) \to \mathsf{CR}_i^{\mathsf{agree}}$
3: **if** $\mathsf{CR}_i^{\mathsf{agree}} = \perp$,
4: **then** $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$, **exit Round 2**
5: set $\mathsf{DisQual}, \mathsf{CompList} = \emptyset$
6: **for** $j \in \mathbb{R}_1$, **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 1, \sigma_j[2], \mathsf{trans}_j[2]) = 1$
7: **then for** $(k, \Gamma_k) \in \mathsf{trans}_j[2]$
8: // put a newly complained dealer in the list
9: **if** $k \notin \mathsf{DisQual}$, parse $\Gamma_k = (sk_j^{(k)}, \cdot)$
10: $//c_0^{(k)}, c_j^{(k)}$ are what P_i received at line 1 of round 2
11: **if** $\mathsf{PKE.Vrfy}(c_0^{(k)}, c_j^{(k)}, \Gamma_k) = 1 \wedge g^{sk_j^{(k)}} \neq \mathsf{cm}_j^{(k)}$
12: **then** $\mathsf{DisQual} = \mathsf{DisQual} \cup \{k\}$
13: $\mathsf{CompList} = \mathsf{CompList} \cup \{(k, \Gamma_k)\}$
14: $(\mathsf{CR}_i^{\mathsf{agree}}, \mathsf{CompList}) \to \mathsf{trans}_i[3]$
15: sign $\mathsf{FS.Sign}(\mathsf{FS}.sk_i[2], \mathsf{trans}_i[3]) \to \sigma_i[3]$
16: $\mathsf{FS.Update}(\mathsf{FS}.sk_i[2]) \to \mathsf{FS}.sk_i[3]$; **erase** $\mathsf{FS}.sk_i[2]$
17: **if** $\mathsf{ComList} \neq \emptyset$, **then broadcast** $(i, \mathsf{trans}_i[3], \sigma_i[3])$

At the end of Round 3: each P_i **do** :

1: **receive** $\{(j, \mathsf{trans}_j[3], \sigma_j[3])\}_{j\in\mathbb{R}_2}$, for $\mathbb{R}_2 \subset [n]$
2: set $\mathsf{DisQual} = \emptyset$
3: // decide the disqualifed set based on broadcast message
4: **for** $j \in \mathbb{R}_2$
5: parse $\mathsf{trans}_j[3] = (\mathsf{CR}_j^{\mathsf{agree}}, \mathsf{CompList})$
6: **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 2, \sigma_j[3], \mathsf{trans}_j[3]) = 1$
7: $\wedge\, \mathsf{VRF.Vrfy}(rvk_j, \mathsf{rand}, \mathsf{ratio}, \text{“agree”}, \mathsf{CR}_j^{\mathsf{agree}}) = 1$
8: **then for** $(k, \Gamma_k) \in \mathsf{CompList}$
9: // put a newly complained dealer in the list
10: **if** $k \notin \mathsf{DisQual} \wedge \mathsf{PKE.Vrfy}(c_0^{(k)}, c_j^{(k)}, \Gamma_k) = 1$
11: $\wedge\, g^{\Gamma_k.m} \neq \mathsf{cm}_j^{(k)}$
12: **then** $\mathsf{DisQual} = \mathsf{DisQual} \cup \{k\}$,
13: set $\mathsf{Qual} = \mathbb{D}_3 \setminus \mathsf{DisQual}$
14: **output:**
15: $pk = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_0^{(j)}, sk_i = \sum_{j\in\mathsf{Qual}} sk_i^{(j)}$
16: $pk_\tau = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_\tau^{(j)}$, for every $\tau \in [n]$

Figure 1: The Any-Trust DKG construction.

- **End of Round 3.** Nodes finalize the set of disqualified dealers based on received complaint lists (lines 1-12). Following that, they create the public key (shares) and secret key share by aggregating contributions from qualified dealers (lines 14-16).

Complexity analysis. We analyze our DKG protocol's computational and communication costs in Table 2. In our analysis: EXP denotes the exponentiation operation within the group $\mathbb{G}$; We primarily focus on the computationally intensive operations, and hence additive operations, being relatively inexpensive, are excluded from the table. The sizes of a group element, a digital signature, and a VRF credential are denoted by $O(\lambda)$. The notation $s \cdot \mathcal{B}(\ell)$ (or $s \cdot \mathcal{M}(\ell)$) implies that s nodes are each contributing ℓ bits to the broadcast channel (or the multicast channel). We will soon show $\mathcal{B}(n\lambda)$ may only incur constant on-chain cost, therefore the broadcast cost will not be a bottleneck even in a massive scale.

For clarity, we differentiate between two scenarios: (1) Good: Represents the optimistic case where no complaints are raised; (2) Bad: Portrays the worst-case scenario with the maximum number of complaints. For this scenario, we also highlight the additional overhead compared to the Good case.

Table 2: The cost of Any-Trust DKG.

	Good	Bad
Comp. (per node)	$(s+2)n\mathsf{EXP}$	$+4n\mathsf{EXP}$
Broadcast	$s \cdot \mathcal{B}(O(n\lambda))$	$+s \cdot \mathcal{B}(O(s\lambda))$
Multicast	0	$+n \cdot \mathcal{M}(O(s\lambda))$

Security analysis. We establish the security of Any-Trust DKG in the following theorem.

Theorem 1. *The Any-Trust DKG satisfies t-consistency, t-correctness, and $(t,k,T_{\mathcal{A}},T_{\mathsf{sim}})$-oracle-aided algebraic simulatability agasint adaptive adversaries (cf Def.2), with $n \geq 2t+1$, $k \leq s(t+1)$ and $T_{\mathsf{sim}} \leq T_{\mathcal{A}} + O(snt)$, under the DDH assumption in the ROM, and assuming the security of the underlying forward-secure signature scheme. For static adversaries, it further achieves the key-expressability(cf. Def.3).*

Proof. Under DDH assumption in ROM, our building blocks including the VRF and the multi-recipient encryption are secure.

First, we argue the t-consistency. Note that the public key pk and the vector of public key shares are deterministically computed based on the set of qualified dealers which are further determined by the information in the broadcast channel. As all honest users have the same view of the broadcast channel, the t-consistency follows easily.

Then, we show the t-correctness. Recall that $pk = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_0^{(j)}$, and $pk_i = \prod_{j\in\mathsf{Qual}} \mathsf{cm}_i^{(j)}$ for $i \in [n]$. Based on line 10 of round 2, for each $j \in \mathsf{Qual}$, with an overwhelming probability, there is a polynomial $f_j(x) \in \mathbb{Z}_p[X]$ whose degree is up to t, such that $\mathsf{cm}_i^{(j)} = g^{f_j(i)}$. Therefore, define $f(X) = \sum_{j\in\mathsf{Qual}} f_j(X)$, and then it follows that $pk = g^{f(0)}$ and $pk_i = g^{f(i)}$. Meanwhile, every honest $\mathcal{P}_i$ should have $f(i)$. If an honest $\mathcal{P}_i$ does not have $f(i)$, there must exist an index $j \in \mathsf{Qual}$ such that $\mathcal{P}_i$ does not have $f_j(i)$. In this case, $\mathcal{P}$ should follow the protocol description and multicast a verifiable complaint against the dealer j to all other parties. As the verifiable complaints are posted to the broadcast channel by an any-trust group, a verifiable complaint against j must be included. Then, j should be disqualified, which contradicts our assumption that j is in Qual.

For correctness, it remains to show that the set Qual is non-empty. By parameter and the security of VRF, the sampled committee contains at least one honest node with high probability. We argue this honest node will be included in Qual. Particularly, this node shall broadcast an honestly generated transcript which contains valid shares. It is easy to see that the complaints in our system are unforgeable, due to the soundness of proof of decryption. Therefore, this node cannot be disqualified because of this transcript. Moreover, although this node may be corrupted after it sent out the transcript, by the forward security of the underlying signature scheme, the adversary cannot send another message with a valid signature in this round, which means the honest node cannot be disqualified because of post-corruption.

Given its length, the analysis for the oracle-aided algebraic security is presented in Lemma.2, and the analysis for the key expressability is in Lemma.3. □

Lemma 2. *The Any-Trust DKG satisfies $(t,k,T_{\mathcal{A}},T_{\mathsf{sim}})$-oracle-aided algebraic simulatability.*

Proof. By definition, if Π satisfies the oracled-aided algebraic simulatability, then, for every adversary $\mathcal{A}$, there will be an algebraic simulator Sim which can indistinguishably simulate the environment for $\mathcal{A}$. We proceed with the proof by presenting the code of a universal simulator Sim which has black-box access to the adversary $\mathcal{A}$.

On inputs a vector of group elements $\zeta = (g^{z_1}, g^{z_2}, \ldots, g^{z_k})$ for $k = s(t+1)$, Sim can simulate each phase of Π for $\mathcal{A}$ as follows.

SETUP. Sim initializes the set of corrupted parties $\mathcal{C} = \emptyset$, the set of honest parties $\mathcal{H} = \{\mathcal{P}_i\}_{i\in[n]}$, and a table $\mathsf{RO}_{\mathsf{hist}} = \emptyset$ to record the query history of the random oracle. Then, it follows the protocol specifications to generate the public parameters and key pairs for all honest users. It answers the adversary's queries as follows.

- **Corruption queries.** When $\mathcal{A}$ asks to corrupt the party $\mathcal{P}_i$, Sim first checks if $|\mathcal{C}| \leq t$. If the check fails, it ignores this query; otherwise, return the secret keys of $\mathcal{P}_i$, and update the sets $\mathcal{H} = \mathcal{H}\backslash\{\mathcal{P}_i\}$ and $\mathcal{C} = \mathcal{C} \cup \{\mathcal{P}_i\}$.

- **Random oracle queries.** When $\mathcal{A}$ queries the random oracle with an input x, Sim checks if x has been asked before. If there is a record of (x, output_x) in $\mathsf{RO}_{\mathsf{hist}}$, return output_x; otherwise, uniformly sample output_x, add (x, output_x) to $\mathsf{RO}_{\mathsf{hist}}$, and return output_x.

Round 1. For every honest party $\mathcal{P}_i \in \mathcal{H}$, Sim runs the *Self-Election* procedure using $\mathcal{P}_i$'s VRF secret key. We assume w.l.o.g. there are $s' \leq s$ honest parties being selected and denote the set by $\mathcal{H}_{\mathsf{ele}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_{s'}\}$, where each party has its credential $\mathsf{CR}_{j,\mathsf{deal}}$. Then, Sim simulates the *Commit to secret* procedure on behalf of each $\mathcal{D}_j \in \mathcal{H}_{\mathsf{ele}}$ as follows.

- Denote $\zeta_j = (\zeta_{j,0}, \zeta_{j,1}, \ldots, \zeta_{j,t}) = (g^{z_{(j-1)(t+1)+1}}, g^{z_{(j-1)(t+1)+2}}, \ldots, g^{z_{j(t+1)}})$.
- Generate the commitments $\mathsf{cm}_{j,\tau} = \prod_{\mu \in [0,t]} \zeta_{j,\mu}^{\tau^\mu}$, for every $\tau \in [0,n]$.
- Generate the ciphertext $(c_{j,0}, c_{j,1}, \ldots, c_{j,n})$, where $c_{j,0} = g^{r_j}$ for some $r_j \leftarrow\!\!\$\, \mathbb{Z}_p$, and $c_{j,\tau} \leftarrow\!\!\$\, \{0,1\}^{\lceil \log p \rceil}$ for $\tau \in [n]$.
- Broadcast $(\mathsf{CR}_{j,\mathsf{deal}}, \mathsf{cm}_{j,0}, \ldots, \mathsf{cm}_{j,n}, c_{j,0}, \ldots, c_{j,n})$.

Sim needs to answer the queries from the adversary. For the random oracle queries made before broadcast and the corruption queries, Sim can respond as it does in the SETUP phase. We discuss its strategy for answering random oracle queries that are made after the broadcast below.

- **Random oracle queries.** Before answering any random oracle queries at this stage, Sim first calculates a matrix of group elements

$$\gamma = \begin{pmatrix} \gamma_{1,1} & \gamma_{1,2} \cdots & \gamma_{1,n} \\ \gamma_{2,1} & \gamma_{2,2} \cdots & \gamma_{2,n} \\ \vdots & \vdots & \vdots \\ \gamma_{s',1} & \gamma_{s',2} \cdots & \gamma_{s',n} \end{pmatrix},$$

  where each $\gamma_{j,\tau} = pk_\tau^{r_j}$ for $j \in [s']$ and $\tau \in [n]$, pk_τ is the encryption public key of $\mathcal{P}_\tau$, and r_j is the randomness used in encryption by Sim when simulating $\mathcal{D}_j$. Sim checks if any $\gamma_{j,\tau}$ has been asked before and aborts if one is in the query history. Otherwise, continue.

  When $\mathcal{A}$ queries a message x, Sim performs as follows.

  If $x \neq \gamma_{j,\tau}$ for any j and τ, proceed as what it did in the DEAL phase.

  If $x = \gamma_{j,\tau}$ for some j and τ, Sim first queries the oracle $\mathsf{DL}_g(\cdot)$ with $\mathsf{cm}_{j,\tau}$ and its representation $(\tau^0, \tau^1, \ldots, \tau^t)$ over ζ_j. Sim will receive $\xi_{j,\tau}$ from the oracle. Then, it sets $\mathsf{output}_{\gamma_{j,\tau}} := c_{j,\tau} \oplus \xi_{j,\tau}$, records $(\gamma_{j,\tau}, \mathsf{output}_{\gamma_{j,\tau}})$ into $\mathsf{RO}_{\mathsf{hist}}$, and returns $\mathsf{output}_{\gamma_{j,\tau}}$ to $\mathcal{A}$.

Other rounds. Sim simulates the behavior of honest parties by following the specifications of the protocol. The queries from $\mathcal{A}$ are answered in the same way as Sim did in the DEAL phase.

Let Qual_C be the set of qualified nodes which are corrupted before the **Round 1**, and $\mathsf{Qual} = \mathsf{Qual}_C \cup \mathcal{H}_{\mathsf{ele}}$. For every $j \in \mathsf{Qual}_C$, the dealer must have distributed its secret shares to honest nodes; otherwise, it will be disqualified. As Sim has always controlled more than $t+1$ honest participants, it can recover the secret key sk_j w.r.t. pk_j for every $j \in \mathsf{Qual}_C$. Therefore, Sim can output the algebraic representation for the public key as:

$$pk = \prod_{j \in \mathsf{Qual}} pk_j = g^{\sum_{j \in \mathsf{Qual}_C} sk_j} \prod_{j \in [s']} g^{z_{(j-1)(t+1)+1}}.$$

Now, we argue that the simulator specified above satisfies the requirements of oracle-aided simulatability. First, it is easy to verify that the running time of Sim is $T_{\mathcal{A}} + O(snt)$.

Then, we show that $\mathsf{view}_{\mathcal{A},y,\Pi}$ and $\mathsf{view}_{\mathcal{A},y,\Pi}$ are identical, under the condition that Sim never aborts during the simulation. Specifically, from the point of $\mathcal{A}$'s view, the commitment sequence outputted by an honest party $\mathcal{D}_j$ is a commitment to the polynomial $f_j(x) = \sum_{\tau=0}^{n} z_{(j-1)(t+1)+\tau+1} x^\tau$. Note that the input group elements of Sim is uniformly sampled, and thus the distribution of $f_j(x)$ is also uniform, which is identical to that in the real experiment. Moreover, in the random oracle model, the distribution of ciphertexts simulated by Sim is also identical to the real distribution, as for every $pk_\tau^{r_j}$ that has been issued to the random oracle, which means that $\mathcal{A}$ can decrypt the ciphertext $c_{j,\tau}$, it follows that

$$c_{j,\tau} = \mathsf{Hash}(pk_\tau^{r_j}) \oplus f_j(\tau).$$

Next, we argue that Sim only aborts with a negligible probability. When Sim aborts, $\mathcal{A}$ must have queried the random oracle with some $x = \gamma_{j,\tau}$ before seeing the broadcast messages. However, $\gamma_{j,\tau} = pk_\tau^{r_j}$ is a uniformly random group element, as r_j is uniformly chosen from $\mathbb{Z}_p$ and completely independent of $\mathcal{A}$'s view before g^{r_j} is broadcasted. Therefore, $\mathcal{A}$ has negligible probability if outputting $pk_\tau^{r_j}$.

Then, we show that Sim has made at most $k-1$ queries to the $\mathsf{DL}_g(\cdot)$ oracle. Recall that Sim makes a query to $\mathsf{DL}_g(\cdot)$ whenever $\mathcal{A}$ queries the random oracle with a message x which is equal to some $\gamma_{j,\tau}$. We note that under the DDH assumption, $\mathcal{A}$ can output $\gamma_{j,\tau} = pk_\tau^{r_j}$ only when $\mathcal{A}$ has corrupted the party $\mathcal{P}_\tau$ (and thus can compute $\gamma_{j,\tau} = (g^{r_j})^{sk_\tau}$), except a negligible probability. As $\mathcal{A}$ can corrupt at most t parties, Sim will query $\mathsf{DL}_g(\cdot)$ at most ts' times, which is smaller than $k-1$.

Finally, we show the simulatability matrix L of Sim is invertible. Without loss of generality, we assume that the adversary has corrupted the parties $\mathcal{P}_1, \ldots, \mathcal{P}_t$, and Sim has made $s't$ queries to $\mathsf{DL}_g(\cdot)$ for simulating the queries from the adversary. For ease of analysis, we let Sim make some dummy queries such that the representations of all the queries are gonna form a square matrix of order $s(t+1)$. Specifically, Sim makes the following extra queries:

$$g^{z_{s'(t+1)+1}}, g^{z_{s'(t+1)+2}}, \ldots, g^{z_{s(t+1)}},$$

and

$$\prod_{\mu \in [0,t]} \zeta_{j,\mu}^{(t+1)^\mu}, \text{ for } j \in [1, s'-1].$$

The number of all queries by Sim is $s't+(s-s')(t+1)+s'-1 = s(t+1)-1$, which is still smaller than k. It is easy to verify the matrix L is invertible. □

Lemma 3. *The Any-Trust DKG satisfies the key-expressability.*

Proof. This proof is similar to the proof for Lemma.2, except we don't need to handle adaptive corruption queries. For any PPT adversary $\mathcal{A}$, we can construct a PPT simulator Sim that takes as input a public key $pk' \in \mathbb{G}$ and simulates the view of $\mathcal{A}$. Assume the sef of corrupted paries is $\{P_i\}_{i\in\mathsf{Corr}}$ for some $\mathsf{Corr} \subset [n]$ and $|\mathsf{Corr}| \leq t$. After sampling the any-trust group, Sim, on behalf of the honest node in the group, creates the following transcript: $\mathsf{cm}_0 = pk'$, $c_0 = g^r$ for some $r \leftarrow\$ \mathbb{Z}_p$; For $i \in \mathsf{Corr}$, $sk_i \leftarrow\$ \mathbb{Z}_p$, $\mathsf{cm}_i = g^{sk_i}$, and $c_i = \mathsf{Hash}(ek_i^r) \oplus sk_i$. For $i \notin \mathsf{Corr}$, cm_i are created by Langrange interpolation in exponent, while c_i are randomly sampled. This transcript is indistinguishable with an honestly generated one in the view of $\mathcal{A}$, and cannot be disquafied. For every other transcript with $\mathsf{cm}^{(j)} = pk^{(j)}$ which is eventually included in the qualified set, Sim can know the secret key $sk^{(j)}$ by reconstructing it from shares held by honest nodes. Note that the final public key is in the form of $pk' \cdot \prod pk^{(j)}$, and the simulator can express it by setting $\alpha = 1$, $sk'' = \sum sk^{(j)}$. □

4 Practical Extended Broadcast Channels

In this section, we introduce a practical extension to the blockchain-based broadcast channel. Despite that it is folklore knowledge that, theoretically, one may throw all messages into the ledger to facilitate a broadcast, but this may incur prohibitive cost in practice, as onchain resources are generally very expensive. Instead, our extension empowers users to broadcast a message of arbitrary length while inscribing only a *constant-size* storage on the blockchain. Crucially, our enhanced broadcast channel retains its original simplicity and modularity. Users can conveniently interact with it using the APIs of well-established infrastructures, including both blockchains and a data dispersal network (DDN) like IPFS [62].

4.1 Building Blocks

We formalize our building blocks. Particularly, for simplicity, we model a blockchain as a public bulletin board (PBB) which allows users to post and retrieve data.

Public Bulletin Board. We follow the model of PBB presented in [46] and extend it to support *keyword*-based retrival. Formally, a user can interact with PBB via the following queries.

- $\mathsf{getCounter}() \to t$. It returns the current counter value t.
- $\mathsf{post}(\mathsf{kw}, v) \to t$. On receiving value v along with a keyword kw, it increments the counter value by 1 to t, stores (t, kw, v), and responses t.
- $\mathsf{retrieve}(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw}) \to \{(v_i, t_i)\}$. It returns all pairs of (v_i, t_i), such that $t_{\mathsf{start}} \leq t_i \leq t_{\mathsf{end}}$ and kw is their keyword.

We care about the storage cost of PBB. For a user posting ℓ bits to the PBB, we denote the cost as $\mathcal{PB}(\ell)$.We assume that a PBB satisfies the *validity* and *agreement*.

VALIDITY. Assume an honest user posted (v, kw) to the PBB and received t. Then, every honest user who retrives with $(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw}')$ such that $t_{\mathsf{start}} \leq t \leq t_{\mathsf{end}}$ and $\mathsf{kw}' = \mathsf{kw}$ will receive a sequence of value/counter pairs containing (v, t).

AGREEMENT. If an honest user retrieving with $(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw})$ when $\mathsf{getCounter}() \geq t_{\mathsf{end}}$ receives a sequence of value/counter pairs S, then every honest user retrieving with $(t_{\mathsf{start}}, t_{\mathsf{end}}, \mathsf{kw})$ will receive the same S.

It is rather straightforward to use PBB as a broadcast channel by simply posting a broadcast message into the PBB. The authenticity can be established with standard digital signatures in the PKI model.

Data Dispersal Network. A data dispersal network (DDN) provides a platform where one can provision a data block for others who may need it. Comparing with standard multicast which is also for data dissemination, DDN saves communication cost when there are multiple nodes providing the same datablock. Assuming there are m receivers out of n potential receivers, and there are k data provider for a datablock of ℓ bits. Through multicast, every sender needs to send their data to every potential receiver, incuring the communication cost of $k \cdot \mathcal{M}(\ell) = O(kn\ell)$. In contrast, through DDN, each receiver receives exactly one copy of data, incurring totoal communication cost of $O(m\ell)$ which is smaller than $\mathcal{M}(\ell)$.

In principle, we can either use an erasure-code-based information dispersal protocol [54] or practical infrastures like IPFS [62] to instantiate a DDN. In this work, we focus on the IPFS-based instantiation as it comes easier to implement (given IPFS already exists) and model it with the following two queries which might be specific to the instantiation.

- register: on receiving a node ID nid and a block ID bid (which is the hash value of the data), it checks whether bid has been registered. If not, add a new entry $(\mathsf{bid}, \mathsf{nid})$; otherwise, it appends nid to the existing entry with bid.
- retrieve: on receiving a block ID bid, it returns the associated datablock v, by orchestrating the data flow from candidate providers.

We assume as long as there is an honest data provider who has registered bid and remains active, everyone can retrieve the data block with bid. We denote the cost of registering for s data blocks as $\mathcal{R}(s)$.

4.2 Our Extended Broadcast Channel

A strawman and our intuition. A naive approach to broadcasting a sizeable data block involves posting its ID, denoted as bid, on the PBB while simultaneously registering both bid and the sender's ID (nid) on the DDN. However, this methodology cannot guarantee agreement. Specifically, a malicious sender has the capability to selectively deny some retrieval requests on the DDN. Moreover, an adaptive adversary, upon observing the bid on the PBB, can corrupt the sender, subsequently rendering the data inaccessible on the DDN.

Round 1: each sender $S_j(v_j)$ **do**:

compute the block ID: $\mathsf{Hash}(v_j) \to \mathsf{bid}_j$

post $\mathsf{PBB.post}(\mathsf{kw}, \mathsf{bid}_j), \mathsf{kw} := (\mathsf{sid}||\mathsf{send})$; **multicast** v_j

Round 2: each receiver P_i **do**:

$\mathsf{PBB.getCounter}() \to t_1'$

// assume the index set of senders is $\mathbb{J}$

$\mathsf{PBB.retrieve}(t_0', t_1', \mathsf{sid}||\mathsf{send}) \to \{(\mathsf{bid}_j, t_j)\}_{j\in\mathbb{J}}$

receive multicast messages: $\{v_j'\}_{j\in\mathbb{J}}$

for $j \in \mathbb{J}$: **if** $\mathsf{Hash}(v_j') = \mathsf{bid}_j$, **then** $\mathsf{valid}_j = 1$; **else** $\mathsf{valid}_j = 0$

$\mathsf{VRF.Sortition}(rvk_i, rsk_i, \mathsf{rand}, \text{“check"}, \mathsf{ratio_{hm}}) \to \mathsf{CR}_i$

if $\mathsf{CR}_i \neq \perp$

then $\mathsf{PBB.post}(\mathsf{kw}', \mathsf{CR}_i||(\mathsf{valid}_j)_{j\in\mathbb{J}}), \mathsf{kw}' := (\mathsf{sid}||\mathsf{check})$

Round 3: each receiver P_i (with node id nid_i) **do**:

$\mathsf{PBB.getCounter}() \to t_2'$

$\mathsf{PBB.retrieve}(t_1', t_2', \mathsf{sid}||\mathsf{check}) \to \{\mathsf{CR}_k||(\mathsf{valid}_j^{(k)})_{j\in\mathbb{J}}\}_{k\in\mathbb{K}'}$

verify every CR_k, and obtain the valid set $\mathbb{K} \subset \mathbb{K}'$

for $j \in \mathbb{J}$: **if** $\sum_{k\in\mathbb{K}} \mathsf{valid}_j^{(k)} \geq \frac{|\mathbb{K}|}{2} + 1$

then $\mathsf{final}_j = 1$; **else** $\mathsf{final}_j = 0$

for $j \in \mathbb{J}$, **if** $\mathsf{final}_j = \mathsf{valid}_j = 1$, **then** $\mathsf{DNN.register}(\mathsf{nid}_i, \mathsf{bid}_j)$

At the end of Round 3: each receiver P_i **do** :

for $j \in \mathbb{J}$ s.t. $\mathsf{valid}_j = 0$:

if $\mathsf{final}_j = 1$, **then** $\mathsf{DNN.retrieve}(\mathsf{bid}_j) \to v_j$; **else** $v_j = \perp$

output $(v_j)_{j\in\mathbb{J}}$

Figure 2: Our extended broadcast channel

To address these security vulnerabilities, we suggest using DDN and PBB together in a smarter way. Recognizing the potential threat of adaptive corruption, the sender directly multicasts the datablock to all receivers while posting the block ID bid into the PBB. Importantly, this process does not induce additional overhead compared with the DDN-based dissemination, since there is only one provider and all receivers will require the data block. To achieve agreement, an honest majority committee is sampled, which subsequently votes to validate the accessibility of the data block against the advertised bid. In scenarios where the majority of the committee members vouch for the data block's availability, all receivers who successfully received the data block are then prompted to register on the DDN. This ensures that any receivers who failed to receive the data through multicast will be able to retrieve the it from DDN.

Protocol details. We assume the PKI setup as well as the setup for the VRF-based sortition, such that everyone in the group gets to know others' verifications keys w.r.t. a digital signature scheme and VRF. A ratio $\mathsf{ratio_{hm}}$ is also determined in the setup, which ensures with high probability that the sampled committee will contain an honest majority. Moreover, we assume every message has been signed by the sender. Besides that, a session id sid and an initial counter t_0' are supposed to be known to everyone in the group and can be used to retrieve related messages from the PBB. We w.l.o.g. describe our protocols in a batch manner, *i.e.*, there can be multiple senders, as this is the situation of our DKG protocol. We elucidate our design in Fig.2.

Complexity analysis. Assume there are s senders, and each of them broadcasts a message of ℓ bits to the group with n nodes. The communication cost of our extended broadcast channel is

$$s \cdot \mathcal{B}(\ell) = s\mathcal{PB}(\lambda) + O(sn\ell) + c\mathcal{PB}(\lambda + s) + n\mathcal{R}(s),$$

where λ denotes the security parameter (*i.e.*, the size of a digest, the output length of a VRF, *e.t.c.*), $s\mathcal{PB}(\lambda)$ is caused by that s senders post their digests into the PBB, $O(sn\ell)$ is caused by that the senders multicast their message and the receivers retrieve from a DDN, $c\mathcal{PB}(\lambda + s)$ is caused by the selected committee members vote for the broadcast status, and $n\mathcal{R}(s)$ is caused by that the honest parties register to the DDN. Now, the onchain storage cost is *independent of* ℓ.

Security analysis. We establish the security of our extended broadcast channell in the following lemma.

Lemma 4. *Assume the underlying PBB satisfies validity and agreement, and the DDN guarantees the data block can be retrieved when there is an honest and active provider. The protocol in Fig.2 satisfies the validity and agreement.*

Proof. Our construction satisfies both the validity and agreement. Regarding validity, in our protocol, when the sender is honest, every honest receiver can receive the message v from the multicast channel and retrieve the digest from the PBB. Then, in round 2, selected honest committee members would vote for this broadcast (by setting and posting $\mathsf{valid} = 1$), such that the final status of this broadcast will be 1, and honest nodes can decide on v.

Regarding agreement, note that whether $v = \perp$ is determined by the votes on PBB. Therefore, if an honest receiver

decides on $v = \perp$, everyone will do the same thing. The potential chance causing disagreement is that when an honest receiver decides on $v \neq \perp$, some receiver cannot successfully retrieve v from the DDN. Below we show this case is unlikely to happen.

Assume that the adversary is allowed to corrupt at most t participants among all the n participants, and the VRF-based sortition at round 2 will yield a committee $\mathcal{C}$ of $c = 2t' + 1$ participants. As the parameter is configured to guarantee the honest majority of the elected committee, it implies that, for any subgroup A whose size is not greater than t, the following probability is very small:

$$\Pr[|Z| \geq t' + 1 : Z = A \cap \mathcal{C}].$$

Now, we consider the group B of nodes that are, before the election, either corrupted nodes or honest nodes that have received v. In the case that there are $t' + 1$ votes endorsing the availability of v, it holds that $|B \cap \mathcal{C}| \geq t' + 1$, which implies the probability $\Pr[|B| \leq t]$ is small. Therefore, the adversary cannot corrupt all nodes in B even after knowing the committee $\mathcal{C}$. It follows that there is always at least one honest node that has received v and provision it to the DDN, such that everyone can retrieve the data from the DDN, and can agree on the value v. □

5 Sub-ID Allocation for the Weighted Setting

In this section, we present a simple-yet-effective sub-ID allocation mechanism that dramatically reduces the number of required sub-IDs.

Qualified allocation. The traditional sub-ID allocation method ensures that the proportion of sub-IDs held by honest participants is equal to the proportion of an honest participant's weights, which we call a *perfect* allocation. However, we notice a gap between the usual assumption on the honest participant's weight ratio, which is typically assumed to be more than 2/3 due to other components of the system, and the honest ratio needed in threshold cryptography, which is usually just above 1/2. Therefore, we consider a lossy-yet-qualified allocation, which guarantees that more than half of the sub-IDs will be issued to honest participants if they have more than 2/3 of the weights[5]. Formally, we have the following definition.

Definition 4 (Qualifed Allocation)**.** Let $W = (w_1, \ldots, w_n)$ be a sequence of positive integers. Let A and B be any partition of the index set $[n]$ (*i.e.*, $A \cup B = [n]$ and $A \cap B = \emptyset$). We say a function $\mathsf{AllocateSubID}(w_1, \ldots, w_n) \to (d_1, \ldots, d_n)$, where d_i's are non-negative integers, is a **qualified allocation** for W, if for every (A, B) s.t.

$$\sum_{i \in A} w_i > 2 \cdot \sum_{i \in B} w_i, \text{ it holds that } \sum_{i \in A} d_i > \sum_{i \in B} d_i.$$

While such a qualified allocation suffices for security, we need to find an allocation method which minimizes the number of all sub-IDs, *i.e.*, $\sum_j d_j$ is as small as possible.

Our method. We start by observing that dividing each w_i by the greatest common division (GCD) leaves the fraction for any index subset A unchanged. This realization provides a straightforward allocation approach: $d_i = \frac{w_i}{\mathsf{gcd}}$. However, if the GCD is small, the total sub-IDs can be vast.

A viable approach is to modify each w_i to w'_i so the new sequence $W' = (w'_1, \ldots, w'_n)$ has a substantial GCD. This adjustment might increase some subsets' proportions while reducing others, potentially strengthening the adversary. Still, we determine that any increased power for the adversary remains capped if we limit the total adjustments. Specifically, if $\sum_{i \in [n]} w_i = 3t + 1$ for a positive integer t, then any partition (A, B) over $[n]$ satisfying $\sum_{i \in A} w_i > 2 \cdot \sum_{i \in B} w_i$ will ensure that $\sum_{i \in A} w'_i > \sum_{i \in B} w'_i$, given the inequality:

$$\sum_{i \in A} w'_i - \sum_{i \in B} w'_i \geq \sum_{i \in A} (w_i - \Delta_i) - \sum_{i \in B} (w_i + \Delta_i) \geq 1 \tag{3}$$

Here, $\Delta_i = |w_i - w'_i|$. Such adjustments are termed t**-bounded**. Sub-IDs, d_i, are derived by dividing w'_i by this higher GCD.

Given our objective to minimize $\sum_j d_j$, the goal is to enhance the GCD. To achieve this, we consider a target gcd, defining an adjustment function $f_{\mathsf{gcd}}(w_i) \to w'_i$ as:

$$w'_i = \begin{cases} w_i - (w_i \bmod \mathsf{gcd}), \text{ if } w_i \bmod \mathsf{gcd} < \mathsf{gcd}/2, \\ w_i + \mathsf{gcd} - (w_i \bmod \mathsf{gcd}), \text{ otherwise.} \end{cases} \tag{4}$$

Starting with $\mathsf{gcd} = 1$, we increase it until f_{gcd} is no longer a t-bounded adjustment for W. Utilizing binary search can quickly find a very large gcd. While variations in $(w_1, \ldots, w_n)$ may suggest larger gcd', our found gcd is practically near-optimal. The allocation algorithm is detailed below.

$\mathsf{AllocateSubID}(w_1, \ldots, w_n)$

binary search the largest gcd from 0 to $\max_i w_i$

s.t. f_{gcd} is t-bounded for $(w_1, \ldots, w_n)$

output $(d_i = \frac{f_{\mathsf{gcd}}(w_i)}{\mathsf{gcd}})_{i \in [n]}$

Our sub-ID allocation is a qualified allocation as per Def.4, since f_{gcd} is t-bounded for $(w_1, \ldots, w_n)$. Moreover, for a set of n validators with an arbitrary power distribution, our method only issues at most $2n$ sub-IDs.

Lemma 5. *Given any sequence $W = (w_i)_{i \in [n]}$ with $\sum_{i \in [n]} w_i = 3t + 1$ for some integer t, let $(d_1, \ldots, d_n)$ be the output of our*

[5] While our discussion primarily centers on the gap between 2/3 and 1/2, the underlying concept can be effortlessly extended to address other thresholds or scenarios.

Table 3: Comparison with Swiper/Dora

Systems	# Parties	#Total Weights	[23]	Ours
Aptos [5]	104	8.4708×10^8	27	34
Tezos [59]	382	6.7579×10^8	75	77
Filecoin [29]	3700	2.5242×10^{19}	1895	1688
Algorand [18]	42920	9.7223×10^9	373	301

AllocateSubID. *It follows that*

$$\sum_{i\in[n]} d_i \leq \frac{4t+1}{\lfloor 2t/n \rfloor},$$

which is around $2n$ *when* $n \ll t$.

Proof. Let $\mathsf{gcd} = \lfloor 2t/n \rfloor$. It is easy to see that $(w'_1,\ldots,w'_n)$ outputted by $f_{\mathsf{gcd}}(w_1,\ldots,w_n)$ and $(w_1,\ldots,w_n)$ are bounded by $n \cdot \lfloor 2t/n \rfloor/2 = t$. Let $d_i = \frac{w'_i}{\mathsf{gcd}}$. It holds that $\sum_{i\in[n]} d_i = \frac{\sum_{i\in n} w'_i}{\lfloor 2t/n \rfloor} \leq \frac{4t+1}{\lfloor 2t/n \rfloor} \approx 2n$. □

Comparison with Swiper/Dora [23]. We notice a concurrent work, Swiper/Dora [23], which also addresses the imparity between conventional threshold cryptography and the weighted setting. In Table 3, we compare our method and theirs for validator sets across various PoS systems. The comparison is under the same condition, *i.e.*, ensuring more than 1/2 sub-IDs are allocated to honest parities with more than 2/3 weights. The result shows our method issues fewer sub-IDs to large sets of validators, such as Algorand and Fielcoin.

6 Application to All-hands Checkpointing into Bitcoin

In this section, we delineate how our DKG yields the first realization of Pikachu's *all-hands* checkpointing vision [6] that involves all validators in the whole blockchain network, e.g., Filecoin, that has 3700 of them, with various mining power.

6.1 The Blueprint of Pikachu

Long-range attacks against PoS blockchain. Unlike proof-of-work chains, block creation in PoS systems is both costless (in terms of physical resources like energy) and timeless (unconstrained by time limits), which enables adversaries to easily fork a chain. Existing PoS chains prevent from malicious forking by punishing misbehavioured validators. However, an attacker can choose to present the fork chain after all its stakes have been withdrawn, thus free of being slashed, What is worse, a later coming client may not be able to decide the canonical chain among the forks.

Securing PoS with Bitcoin checkpointing. A few works [6, 58] have shown that long-range attacks can be effectively mitigated by creating checkpoints of the PoS chain on a PoW chain, such that a late comming client can distinguish the canonical chain among forks. Pikachu illustrates a threshold signature-based checkpointing mechanism. At a high level, the lifetime of the PoS system is divided into multiple epochs, and checkpoints are supposed to be created per epoch. At every epoch i, a configuration $C_i = \{(\mathcal{V}_{i,j}, w_{i,j})\}_{j\in[n_i]}$ for some integer n_i, which is the set of all validators $\{\mathcal{V}_{i,j}\}_{j\in[n_i]}$ with their weights $\{w_{i,j}\}_{j\in[n_i]}$, is associated with a public key Q_i (w.r.t. Schnorr signature scheme) which can serve as a Bitcoin address, while the secret key of Q_i is secretly shared among C_i. At epoch $i+1$, validators in C_i will jointly create a Bitcoin transaction which transfer all assets on Q_i to Q_{i+1}, the address belong to the current configuration C_{i+1}; This transaction is the checkpoint. We elucidate their design with the following three algorithms/protocols[6].

- $\mathsf{AllocateSubID}(C) \to \{d_j\}_{j\in[n]}$.The sub-identity allocation algorithm takes input as a configuration $C = \{(\mathcal{V}_j, w_j)\}_{j\in[n]}$ and determines the number of sub-identities d_j for each $\mathcal{V}_j$ according to their weight w_j.
- $\mathsf{DKG}(\{(\mathcal{V}_j, d_j)\}_{j\in[n]})$. The validators in C run a DKG protocol, while each sub-identity is viewed as an independent participant. Therefore, each validator $\mathcal{V}_j$ obtains d_j pairs of $(pk_{j,z}, sk_{j,z})_{z\in[d_j]}$, and all validators obtain the same public key $Q = pk$ and the list of public key shares $\vec{pk} = (pk_{j,z})_{j\in[n],z\in[d_j]}$.
- $\mathsf{CreateCKP}(C_i, \mathsf{ckp}, \mathsf{PreAdd}, Q_{i+1}) \to \mathsf{TX}$. At the epoch $i+1$, assume that validators in C_{i+1} have generated the public key Q_{i+1}, the digest of PoS block to be checkpointed is ckp, and the address of the last checkpointing Bitcoin transaction is PreAdd. Then, the validators in C_i inovke a Threshold Schnorr protocol to sign a Bitcoin transaction TX with the follwoing information.

  $$\{\mathsf{Input} : \mathsf{PreAdd}; \mathsf{Output} : Q_{i+1}; \mathsf{OP\_Return} : \mathsf{ckp}\}.$$

  Once the transaction has been properly signed, every validator should disseminate it to the Bitcoin network.

With checkpoints on Bitcoin, it is rather straightforward for a late-coming user to decide which fork is the canonical chain, when the user is provided with a blocktree of finalized PoS blocks. Specifically, the user first synchronizes with Bitcoin blockchain. Then, it finds the initial checkpoint transaction and builds a chain of transactions following the initial transaction. Next, it obtains the digest ckp from the latest checkpoint transaction, and decides the fork with the a block whose digest is ckp as the canonical chain. Moreover, while other approaches like key-evolving forward-secure signatures [18, 21] may also mitigate long-range attacks, the

[6] Slightly different from their original descirption where the PoS digest is embeded into the Bitcoin address, we choose to put it in OP_RETURN for simplicity.

Table 4: Checkpointing cost per annum. in USD.

#Parties	Babylon(o.)	Babylon (s.)	Ours
2^7(Cosmos)	4,304.7	86,093.3	2,152.3
2^{10}(Polkadot)	6,457.0	129,140.0	2,152.3
2^{12}(Filecoin)	17,218.7	344,373.1	2,152.3

*Calculation based on the Bitcoin price on on Oct. 16, 2023: 0.000273 USD per Satoshi.

checkpointing mechanism enjoys a unique advantage of ensuring malicious validators are always slashable. We defer a detailed discussion to Sect.6.4.

6.2 Realizing Pikachu with Any-Trust DKG

Pikachu only demonstrated a proof-of-concept prototype with 21 participants, due to the inefficiency of their underlying DKG scheme. Meanwhile, as they instantiated the threshold Schnorr signature with FROST [47] which relies on a coordinator, there may be a single point of failure. Our Any-Trust DKG can realize Pikachu efficiently and securely.

Sub-ID allocation. Our optimized sub-ID allocation algorithm in Sect.5 issues fewer sub-IDs to validators. We consider a snapshot of Filecoin's validator distribution[7], which has 3700 validators with total mining power of around 22 EB, while the power unit is 32KB. The standard method may issue around 674 trillion sub-IDs. In contrast, our method identifies that 13 PB can be a good GCD and only 1688 sub-IDs need to be issued, significantly reducing the scale of the problem.

Apply Any-Trust DKG. Then we apply our Any-Trust DKG for the 1688 sub-IDs. We set $\mathsf{ratio}_{\mathsf{at}} = 20/1688$, which guarantees the committee has at least one good node with high probability of $1-5\times10^{-9}$. Then, only around 1.6 MB data needs to be broadcasted. It takes each node a few seconds to finish computation, even facing the maximum number of complaints.

Checkpointing with non-interactive threshold Schnorr signature. To sign the checkpoint transaction, we adopt the GJKR protocol [34], which does not require a coordinator and is thus free of single-point of failures. The GJKR protocol involves a DKG as its subroutine for generating the nonce and follows a non-interactive phase where every signer can locally compute its signature share (or called a partial signature). GJKR was believed to be inadequate for large-scale deployment due its DKG subroutine, which however is no longer a bottleneck with our any-trust DKG. Since our DKG is key-expressable (cf. Def.3 and [41]), the static security of the resulting scheme direcly follows the recent result in [56]. While there is no direct adaptive attack, we leave a fully analysis for adaptive security as a future study.

[7] https://filfox.info/en/ranks/power

6.3 Comparison with Babylon Checkpointing

Overview of Babylon. Babylon [58] is a recently proposed checkpointing scheme that does not use DKG and threshold signature. Instead, it employs the following approach: (1) All validators sign the digest of the PoS block to be checkpointed. (2) One honest validator collects and aggregates enough signatures (using the BLS aggregatable signature scheme [12]) and publishes a Bitcoin transaction with the OP_RETURN code. This transaction contains the digest, the aggregated signature, and the public keys.

Comparison of Bitcoin Transaction Fees. It's important to note that for n validators, at least n bits are needed to encode the public key list. A Bitcoin transaction allows 80 bytes with OP_RETURN, which means the number of Bitcoin transactions per checkpoint grows linearly with the number of validators. Taking into account the fixed cost for storing the aggregated signature and checkpointing identifier, the number of Bitcoin transactions for a Babylon checkpoint can be calculated as $\#\mathsf{Bitcoin\ Tx}_{\mathsf{Babylon}} = 1+\lceil\frac{n+32}{640}\rceil$. Moreover, since it assumes an honest validator to create the checkpointing transaction, it might have a single point of failure. This issue can be resolved by sampling a committee that includes at least one honest validator for creating Bitcoin transactions. For the more secure version of Babylon, the number of Bitcoin transactions per checkpoint would increase by a factor of the statistical security parameter s, i.e., $\#\mathsf{Bitcoin\ Tx}_{\mathsf{secure-Babylon}} = s+s\cdot\lceil\frac{n+32}{640}\rceil$. For $s=20$ and $n=2^{12}$, we have

$$\#\mathsf{Bitcoin\ Tx}_{\mathsf{Babylon}} = 8, \text{ while } \#\mathsf{Bitcoin\ Tx}_{\mathsf{secure-Babylon}} = 160.$$

In comparison, our approach (Pikachu) only requires 1 Bitcoin transaction for each checkpoint, and it is naturally free of single points of failure.

We compare the Bitcoin transaction fees for checkpointing per annum in Table 4. Babylon(o.) refers to the original checkpointing mechanism in Babylon, which has a single point of failure. Babylon(s.) refers to the secure version where a small committee (we set $s=20$) posts the checkpointing transactions into Bitcoin. Following [58], we consider the checkpoint transactions to be created hourly. We assume, without loss of generality, that each Bitcoin transaction has 300 bytes, the transaction fee is 3 Satoshis per byte (as suggested in [58]), and the price of a Satoshi is 0.000273 USD [8]. We evaluate the cost for PoS chains with different numbers of validators: 2^7 validators for small-scale PoS chains (like the ones in Cosmos [19]), 2^{10}validators for moderate-scale chains (like Polkadot [53]), and 2^{12} validators for large-scale chains (like Filecoin [29]).

[8] updated on Oct. 16, 2023, from https://coincodex.com/crypto/satoshi-sats/

6.4 Security of Checkpointing

This paradigm has been thoroughly analyzed in [6]. It considers an efficient adversary $\mathcal{A}$, which at each epoch i can corrupt all validators in previous configurations $\{C_j\}_{j<i-L}$ and a fraction of validators up to f in "recent" configurations $\{C_j\}_{i-L<j\leq i}$, for some parameter L such that the checkpoint transaction for epoch i_0 will be confirmed in Bitcoin by epoch i_0+L. Such an adversary can mount long range attacks by using the previous secret keys to forge another validate-looking chain (called a long-range attack chain). However, since the Bitcoin blockchain has recorded transactions which transferred all assets from previous addresses $\{Q_j\}_{j<i-L}$, $\mathcal{A}$ cannot create valid checkpoints using secret keys of $\{Q_j\}_{j<i-L}$. Therefore, a bootstrapping client can decide the canonical chain with Bitcoin checkpoints. We summarize their results in the following theorem.

Theorem 6 ([6]). *Assume both Bitcoin blockchain and the PoS chain satisfy consistency, chain growth, and chain quality (as defined in [33]). Assume the Threshold Schnorr signature satisfies unforgeability and robustness under the DKG protocol against $\mathcal{A}$ corrupting up to t sub-identities, and* AllocateSubID *allocates at most t sub-identies to $\mathcal{A}$. Then, the checkpointing mechanism satisfies the following properties.*

- *Safety. $\mathcal{A}$ cannot produce any valid checkpointing transactions for long range attack chains.*
- *Liveness. $\mathcal{A}$ cannot stop the checkpoints from happening.*

On Slashable Safety. Babylon claims the slashable safefy. Specifically, for a PoS system with $3t+1$ units of stake, validators with at least t units should become slashable in the view of all honest validators, whenever there is a safety violation. Many PoS systems offer slashable safety against *short-range* attacks by locking validators' stake for a period and slashing one's stake once a proof of security violation is presented. However, long-range attackers can evade being slashed by publishing the attack chain after withdrawing their stakes from the canonical chain.

It has been proved in [58] that slashable safety against long-range attacks is impossible without external trust. With this result, [58] also shows that other approaches for mitigating long-range attacks such as key-evolving signatures [8, 18] cannot provide slashable safety. Nonetheless, leveraging the Bitcoin blockchain as external trust can certainly bypass this impossibility. Assuming that checkpoints for the canonical PoS chain has been properly posted on the Bitcoin blockchain, the attacker cannot present an attack chain which diverges from the canonical chain before the latest checkpoint. In this case, the attacker must have not withdrawn its stakes and thus is slashable.

In the light of above, both ours/Pikachu and Babylon can guarantee slashable safety once the checkpoints have been properly created. Now we turn to examine the case that checkpoints may not be generated correctly. The adversary have the following options (1) not make a checkpoint; (2) make a checkpoint for an ill-formed block; (3) make more than one checkpoints for different well-formed blocks at the same height and hide the block whose checkpoint appears earlier; (4) make a checkpoint for a well-formed block but excluding some valid transactions (for censorship). Babylon introduces an *emergency break* to prevent from (2) and (3). The client can notice these attacks happening and then no longer processes this chain. In case that the adversary refuses to participant in the checkpoint creation, Babylon considered the punishment of inactivity, which enables to remove the inactive valididators. Regarding censorship resistance (4), Babylon proposed a roll-up technique which is orthogonal to the checkpointing mechanism.

In our system, as all checkpoints are in the chain of transactions, the adversary cannot mount the attack of (3). For (2) and (4), we can follow the exact same approach as Babylon does. For the attack of (1), it may be hard to identify who makes the DKG/threshold signing fail. Instead, we require a checkpoint to be made by a certain height of the Bitcoin blochain, and then the client can switch to *emergency* break when it does not find a valid checkpoint by the designated position. In summary, our checkpointing mechanism provides slashable safety, as long as honest clients do not switch to emergency break.

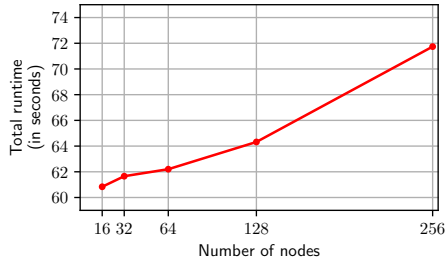
7 Implementation and Evaluation

We implemented our proposed DKG and present the experimental results in this section.

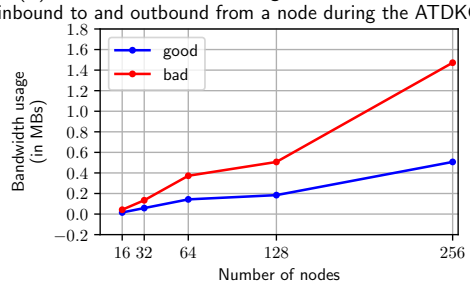
Implementation. We implemented our proposed protocol in Java 8, comprising approximately 1500 lines of code. To facilitate Elliptic Curve operations, we utilized the open-source Java library `mpc4j`[9] and the `Bouncy Castle` library[10]. Given our protocol's primary application in creating checkpoints on Bitcoin, we opted for the secp256k1 curve and SHA-256 for relevant cryptographic operations. Our implementation includes components such as verifiable random function and verifiable multi-recipient encryption, both based on `mpc4j`. It is essential to note that this implementation serves as a proof-of-concept, demonstrating the practicality of our protocol for large-scale deployment, even under the presence of the maximal number of Byzantine nodes. We do not implement forward-secure signatures; however, their cost is marginal and independent of the scale. Throughout our analysis, we set the expected size s of an any-trust group to 20, while the expected size c of an honest-majority group is 296. This configuration ensures that the committee qualifies with a probability of $1-5\times10^{-9}$.

[9] https://github.com/alibaba-edu/mpc4j
[10] https://www.bouncycastle.org/

(A) Worst-case runtime of bad instances, measured by the time difference between the start of the ATDKG and the time the last node outputs keys.



(B) Worst-case bandwidth usage, the amount of data transfers inbound to and outbound from a node during the ATDKG protocol.



(C) Computation time, measured by the time difference between the total time and the communication cost in the worst-case.

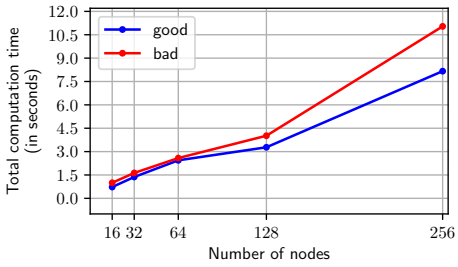


Figure 3: End-to-end Test Results

7.1 End-to-End Implementation

Evaluation Setup. We evaluate our Any-Trust DKG implementation with a varying number of nodes: 16, 32, 64, 128, and 256. Each node is encapsulated within an individual Amazon Web Services (AWS) **t3a.medium** EC2 virtual machines (VM). Each VM has 2 vCPUs and 4 GiB RAM, and runs in Amazon Linux 2023 AMI 2023.2.20231016.0 x86_64 HVM kernel-6.1. All nodes are placed in the same AWS region, and are connected pair-wise, e.g., every two nodes are directly connected. Since the network delay within the same AWS region is almost negligible, we simulate a more realistic delay by employing the Linux command `tc` (traffic control) to introduce an artificial delay of 100 milliseconds for all TCP traffic.

Implementation Remarks. Apart from the nodes that actually run the Any-Trust DKG protocol, an extra node is attached in the network to simulate a blockchain. This blockchain node collaborates with normal nodes and enables broadcasting needed in **Round 1** and **Round 3** of the Any-Trust DKG protocol (See Fig.1). Our implementation of the blockchain allows it to receive packets in a period of 20 seconds, relay packets to other nodes in the next 10 seconds and idle so that the broadcast costs **exactly** 30 seconds. This behaviour mimics a block confirmation that happens in block periodically, and ensures the broadcast can be delivered in 30 seconds. Note that a non-blockchain node is also forced to idle so that the **Round** needing a broadcast can finish in **exactly** 30 seconds, ensuring it is synchronised with the blockchain node.

Evaluation Results. With the evaluation, we aim to demonstrate that our Any-Trust DKG protocol can scale well with the number of nodes and has low computing time and bandwidth usage.

Running Time. We measure the running time of the whole Any-Trust DKG protocol by capturing the time difference between the moment when the communication network is established, and when a node finishes computing the shared public key and its secret share. We take the average of this time difference across all nodes to compute the end-to-end running time of our protocol. The results are in Fig.3.

For $n = 16, 32, 64, 128, 256$, our Any-Trust DKG protocol takes approximately 60 seconds. However, it is notable that the simulated blockchain node will enforce a two timeouts during the execution of the protocol and each lasts for 30 seconds. This implies the actual computation time remains little compared with the total communication time.

Bandwidth Usage. We record the outbound total bandwidth of each node in byte and demonstrate the average results in Fig.3. The key observation is that the bandwidth grows linearly with regards to the size of the group.

7.2 Performance Analysis on Large Scale

While our end-to-end implementation demonstrates that our protocol remains practical when $n = 2^8$, we further tested the computation time of our protocol and estimated the communication cost on larger scales ranging from $n = 2^9$ to $n = 2^{15}$. This range covers the sizes of most PoS chain validators.

On-chain storage cost. We first examine the on-chain storage cost of our protocol, which may be the most crucial part of broadcast cost. With our extended broadcast channel (cf. Sect.4), our protocol just requires the blockchain to store $c + s \approx 312$ normal transactions. Assume each transaction has 300 Bytes, and then our storage cost will be 92.6 KB. In contrast, even in the good case, KZG still has n parties to broadcast messages, which means n transactions are necessary, incurring $300 \cdot n$ Bytes, which can be 9600 KB for $n = 2^{15}$.

We also estimate the total number of bits to be sent via the broadcast channel, a metric independent of the underlying broadcast channel instantiation. We compare our protocol and

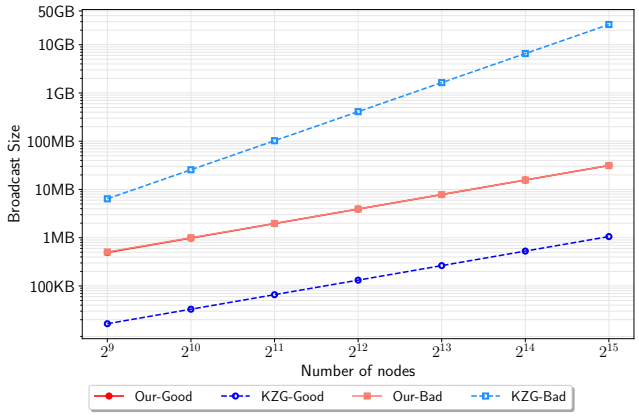


Figure 4: Broadcast Channel Overhead

KZG in terms of it, ranging from $n = 2^9$ to $n = 2^{15}$, considering both the good case and the bad case with the maximal number of complaints. As shown in Figure 4, for our protocol, the costs in the good case and the worst case are very close and grow steadily. For $n = 2^9$, the cost is around 493 KB, while for $n = 2^{15}$, the cost is approximately 31 MB. In contrast, while the good-case KZG has very low broadcast costs, its worst-case costs grow quadratically and would require over 26 GB when $n = 2^{15}$.

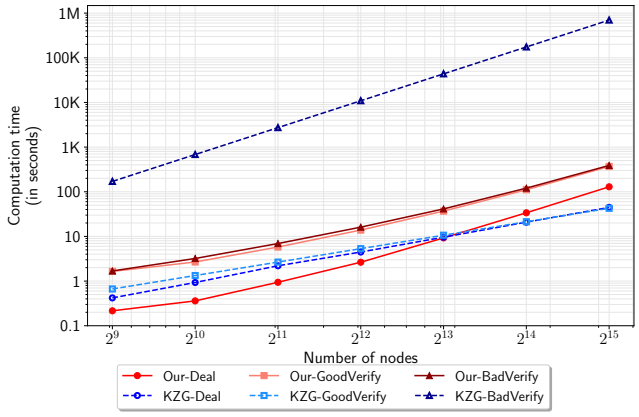


Figure 5: Computation Overhead

Computation time. We conducted tests to measure the computation time for generating a secret-sharing transcript (Deal) and reaching agreement on a qualified set (Verify) in both good case and bad case, on AWS c5a.large (AMD EYPC 7002 CPU with 2 cores and 4 GB RAM). We compared our results with KZG, utilizing the reported findings from [65] for the good case, while estimating the worst-case scenario by assuming that $n^2/2$ shares need to be verified. As illustrated in Fig.5, in the good case, our protocol's performance is comparable to or even better than KZG, despite their programming language (C++) and environment (AWS c5a.24xlarge, AMD EYPC 7002 CPU with 96 cores, and 187 GB RAM) are supposed to be superior to ours. However, in the worst-case scenario, our protocol remains efficient, while KZG becomes infeasible.

Note that our computation time grows faster than KZG's, which we believe is due to the use of a naive implementation of multi-point polynomial evaluation. The complexity of our current implementation is $O(n^2)$ for evaluating an $O(n)$-degree polynomial at $O(n)$ points. In contrast, the implementation in [65] employs an optimized algorithm whose complexity is $O(n \log^2 n)$. However, it is important to highlight that our DKG protocol can benefit from the $O(n \log^2 n)$ polynomial evaluation algorithm as well, and our implementation can be enhanced if a Java implementation for the algorithm becomes available.

Acknowledgements

We thank Marko Vukolić and Alejandro Ranchal-Pedrosa for the helpful discussions. This work was supported in part by Protocol Labs Research Grants under the RFP-012 on Checkpointing Filecoin onto Bitcoin.

References

[1] Makerdao, 2023. https://makerdao.com.

[2] Veramo, 2023. Accessed: Oct. 14, 2023.

[3] Wildleaks, 2023. https://wildleaks.org/.

[4] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., and Stern, G. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *CRYPTO (1)* (2023), vol. 14081 of *Lecture Notes in Computer Science*, Springer, pp. 39–70.

[5] Aptos. https://aptoscan.com.

[6] Azouvi, S., and Vukolic, M. Pikachu: Securing pos blockchains from long-range attacks by checkpointing into bitcoin pow using taproot. *CoRR abs/2208.05408* (2022).

[7] Bacho, R., and Loss, J. On the adaptive security of the threshold BLS signature scheme. In *CCS* (2022), ACM, pp. 193–207.

[8] Badertscher, C., Gazi, P., Kiayias, A., Russell, A., and Zikas, V. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *CCS* (2018), ACM, pp. 913–930.

[9] Bellare, M., Boldyreva, A., Kurosawa, K., and Staddon, J. Multirecipient encryption schemes: How to save on bandwidth and computation without sacrificing security. *IEEE Trans. Inf. Theory 53*, 11 (2007), 3927–3943.

[10] Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., and Reyzin, L. Can a public blockchain keep a secret? In *TCC (1)* (2020), vol. 12550 of *Lecture Notes in Computer Science*, Springer, pp. 260–290.

[11] Benhamouda, F., Halevi, S., Krawczyk, H., Miao, A., and Rabin, T. Threshold cryptography as a service (in the multiserver and YOSO models). In *CCS* (2022), ACM, pp. 323–336.

[12] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the weil pairing. In *ASIACRYPT* (2001), vol. 2248 of *Lecture Notes in Computer Science*, Springer, pp. 514–532.

[13] Bracha, G. Asynchronous byzantine agreement protocols. *Information and Computation 75*, 2 (1987), 130–143.

[14] CANETTI, R., GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. Adaptive security for threshold cryptosystems. In *CRYPTO* (1999), vol. 1666 of *Lecture Notes in Computer Science*, Springer, pp. 98–115.

[15] CASCUDO, I., AND DAVID, B. SCRAPE: scalable randomness attested by public entities. In *ACNS* (2017), vol. 10355 of *Lecture Notes in Computer Science*, Springer, pp. 537–556.

[16] CERULLI, A., CONNOLLY, A., NEVEN, G., PREISS, F., AND SHOUP, V. vetkeys: How a blockchain can keep many secrets. *IACR Cryptol. ePrint Arch.* (2023), 616.

[17] CHAUM, D., AND PEDERSEN, T. P. Wallet databases with observers. In *CRYPTO* (1992), vol. 740 of *Lecture Notes in Computer Science*, Springer, pp. 89–105.

[18] CHEN, J., AND MICALI, S. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci. 777* (2019), 155–183.

[19] COSMOS. https://cosmos.network.

[20] DAS, S., YUREK, T., XIANG, Z., MILLER, A., KOKORIS-KOGIAS, L., AND REN, L. Practical asynchronous distributed key generation. In *SP* (2022), IEEE, pp. 2518–2534.

[21] DAVID, B., GAZI, P., KIAYIAS, A., AND RUSSELL, A. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *EUROCRYPT (2)* (2018), vol. 10821 of *Lecture Notes in Computer Science*, Springer, pp. 66–98.

[22] DAVID, B., MAGRI, B., MATT, C., NIELSEN, J. B., AND TSCHUDI, D. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *CCS* (2022), ACM, pp. 683–696.

[23] DE SOUZA, L. F., AND TONKIKH, A. Swiper and dora: efficient solutions to weighted distributed problems. *CoRR abs/2307.15561* (2023).

[24] DFINITY. Distributed key generation in js, 2019. https://github.com/dfinity-side-projects/dkg.

[25] DNOSIS. Distributed key generation, 2018. https://github.com/gnosis/dkg.

[26] DOLEV, D., AND REISCHUK, R. Bounds on information exchange for byzantine agreement. In *PODC* (1982), ACM, pp. 132–140.

[27] DRAND. A distributed randomness beacon daemon - go implementation, 2023. https://github.com/drand/drand.

[28] FELDMAN, P. A practical scheme for non-interactive verifiable secret sharing. In *FOCS* (1987), IEEE Computer Society, pp. 427–437.

[29] FILECOIN. https://filecoin.io/.

[30] FOUQUE, P., AND STERN, J. One round threshold discrete-log key generation without private channels. In *Public Key Cryptography* (2001), vol. 1992 of *Lecture Notes in Computer Science*, Springer, pp. 300–316.

[31] GAO, Y., LU, Y., LU, Z., TANG, Q., XU, J., AND ZHANG, Z. Dumbo-ng: Fast asynchronous BFT consensus with throughput-oblivious latency. In *CCS* (2022), ACM, pp. 1187–1201.

[32] GAO, Y., LU, Y., LU, Z., TANG, Q., XU, J., AND ZHANG, Z. Efficient asynchronous byzantine agreement without private setups. In *ICDCS* (2022), IEEE, pp. 246–257.

[33] GARAY, J. A., KIAYIAS, A., AND LEONARDOS, N. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)* (2015), vol. 9057 of *Lecture Notes in Computer Science*, Springer, pp. 281–310.

[34] GENNARO, R., JARECKI, S., KRAWCZYK, H., AND RABIN, T. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol. 20*, 1 (2007), 51–83.

[35] GENTRY, C., HALEVI, S., KRAWCZYK, H., MAGRI, B., NIELSEN, J. B., RABIN, T., AND YAKOUBOV, S. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO (2)* (2021), vol. 12826 of *Lecture Notes in Computer Science*, Springer, pp. 64–93.

[36] GENTRY, C., HALEVI, S., AND LYUBASHEVSKY, V. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *EUROCRYPT (1)* (2022), vol. 13275 of *Lecture Notes in Computer Science*, Springer, pp. 458–487.

[37] GENTRY, C., HALEVI, S., MAGRI, B., NIELSEN, J. B., AND YAKOUBOV, S. Random-index PIR and applications. In *TCC (3)* (2021), vol. 13044 of *Lecture Notes in Computer Science*, Springer, pp. 32–61.

[38] GOLDBERG, S., NAOR, M., PAPADOPOULOS, D., AND REYZIN, L. NSEC5 from elliptic curves: Provably preventing DNSSEC zone enumeration with shorter responses. *IACR Cryptol. ePrint Arch.* (2016), 83.

[39] GUO, B., LU, Y., LU, Z., TANG, Q., XU, J., AND ZHANG, Z. Speeding dumbo: Pushing asynchronous BFT closer to practice. In *NDSS* (2022), The Internet Society.

[40] GUO, B., LU, Z., TANG, Q., XU, J., AND ZHANG, Z. Dumbo: Faster asynchronous BFT protocols. In *CCS* (2020), ACM, pp. 803–818.

[41] GURKAN, K., JOVANOVIC, P., MALLER, M., MEIKLEJOHN, S., STERN, G., AND TOMESCU, A. Aggregatable distributed key generation. In *EUROCRYPT (1)* (2021), vol. 12696 of *Lecture Notes in Computer Science*, Springer, pp. 147–176.

[42] HEUER, F., JAGER, T., KILTZ, E., AND SCHÄGE, S. On the selective opening security of practical public-key encryption schemes. In *Public Key Cryptography* (2015), vol. 9020 of *Lecture Notes in Computer Science*, Springer, pp. 27–51.

[43] HUANG, Z., LAI, J., CHEN, W., RAEES-UL-HAQ, M., AND JIANG, L. Practical public key encryption with selective opening security for receivers. *Inf. Sci. 478* (2019), 15–27.

[44] ITKIS, G., AND REYZIN, L. Forward-secure signatures with optimal signing and verifying. In *CRYPTO* (2001), vol. 2139 of *Lecture Notes in Computer Science*, Springer, pp. 332–354.

[45] KATE, A., ZAVERUCHA, G. M., AND GOLDBERG, I. Constant-size commitments to polynomials and their applications. In *ASIACRYPT* (2010), vol. 6477 of *Lecture Notes in Computer Science*, Springer, pp. 177–194.

[46] KIDRON, D., AND LINDELL, Y. Impossibility results for universal composability in public-key models and with fixed inputs. *J. Cryptol. 24*, 3 (2011), 517–544.

[47] KOMLO, C., AND GOLDBERG, I. FROST: flexible round-optimized schnorr threshold signatures. In *SAC* (2020), vol. 12804 of *Lecture Notes in Computer Science*, Springer, pp. 34–65.

[48] LEE, S., MURASHKIN, A., DERKA, M., AND GORZNY, J. Sok: Not quite water under the bridge: Review of cross-chain bridge hacks. In *ICBC* (2023), IEEE, pp. 1–14.

[49] MALKHI, D., AND SZALACHOWSKI, P. Maximal extractable value (MEV) protection on a DAG. In *Tokenomics* (2022), vol. 110 of *OASIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 6:1–6:17.

[50] MILLER, A., XIA, Y., CROMAN, K., SHI, E., AND SONG, D. The honey badger of BFT protocols. In *CCS* (2016), ACM, pp. 31–42.

[51] NAYAK, K., REN, L., SHI, E., VAIDYA, N. H., AND XIANG, Z. Improved extension protocols for byzantine broadcast and agreement. In *DISC* (2020), vol. 179 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 28:1–28:17.

[52] PEDERSEN, T. P. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO* (1991), vol. 576 of *Lecture Notes in Computer Science*, Springer, pp. 129–140.

[53] POLKADOT. https://www.polkadot.network/.

[54] REED, I. S., AND SOLOMON, G. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics 8*, 2 (1960), 300–304.

[55] RUBY.EXCHANGE. How skale solves the front-running problem, 2021. https://blog.ruby.exchange/how-skale-solves-the-front-running-problem/?ref=blog.pantherprotocol.io.

[56] SHOUP, V. The many faces of schnorr. *IACR Cryptol. ePrint Arch.* (2023), 1019.

[57] STEINHOFF, S., STATHAKOPOULOU, C., PAVLOVIC, M., AND VUKOLIC, M. BMS: secure decentralized reconfiguration for blockchain and BFT systems. *CoRR abs/2109.03913* (2021).

[58] TAS, E. N., TSE, D., GAI, F., KANNAN, S., MADDAH-ALI, M. A., AND YU, F. Bitcoin-enhanced proof-of-stake security: Possibilities and impossibilities. In *SP* (2023), IEEE, pp. 126–145.

[59] TEZOS. https://tezos.com.

[60] TOMESCU, A., CHEN, R., ZHENG, Y., ABRAHAM, I., PINKAS, B., GOLAN-GUETA, G., AND DEVADAS, S. Towards scalable threshold cryptosystems. In *IEEE Symposium on Security and Privacy* (2020), IEEE, pp. 877–893.

[61] TOTAL-BLOCKCHAIN. Osmosis will soon be frontrunning mev free, 2022. https://medium.com/@totalblockchainemail/osmosis-will-soon-be-frontrunning-mev-free-b7da89f04ce9.

[62] TRAUTWEIN, D., RAMAN, A., TYSON, G., CASTRO, I., SCOTT, W., SCHUBOTZ, M., GIPP, B., AND PSARAS, Y. Design and evaluation of IPFS: a storage layer for the decentralized web. In *SIGCOMM* (2022), ACM, pp. 739–752.

[63] WOLINSKY, D. I., CORRIGAN-GIBBS, H., FORD, B., AND JOHNSON, A. Scalable anonymous group communication in the anytrust model. In *European Workshop on System Security (EuroSec)* (2012), vol. 4.

[64] YUREK, T., LUO, L., FAIROZE, J., KATE, A., AND MILLER, A. hbacss: How to robustly share many secrets. In *NDSS* (2022), The Internet Society.

[65] ZHANG, J., XIE, T., HOANG, T., SHI, E., AND ZHANG, Y. Polynomial commitment with a one-to-many prover and applications. In *USENIX Security Symposium* (2022), USENIX Association, pp. 2965–2982.