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

Hanwen Feng*, Tiancheng Mai*, and Qiang Tang*

* School of Computer Science
University of Sydney, Australia
{hanwen.feng,tiancheng.mai,qiang.tang}@sydney.edu.au

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 the heavy computation and communication (particularly broadcast) overhead in their adversarial cases. In this paper, we propose a practical DKG for DLog-based cryptosystems, which achieves (quasi-)linear computation and communication per-node cost with the help of a common coin, even in the face of the maximal amount of Byzantine nodes. Moreover, our protocol is secure against adaptive adversaries, which can corrupt less than half of all nodes. The key to our improvements lies in delegating the most costly operations to an *Any-Trust* group together with a set of techniques for adaptive security. 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. Moreover, we present a generic transformer that enables us to efficiently deploy a conventional distributed protocol like our DKG, even when the participants have different *weights*. 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.

Our DKG leads to a fully practical instantiation of Filecoin's checkpointing mechanism, in which *all* validators of a Proof-of-Stake (PoS) blockchain periodically run DKG and threshold signing to create checkpoints on Bitcoin, to enhance the security of the PoS chain. In comparison with the recent checkpointing approach of Babylon (Oakland, 2023), ours enjoys a significantly smaller cost of Bitcoin transaction fees. For 2^{12} validators, our cost is merely 0.4% of that incurred by Babylon's approach.

1 INTRODUCTION

Distributed key generation protocols (DKG) [35, 56] enable a set of participants to jointly generate a public key, and each of them outputs a secret key share. It is a classical topic and the basis of threshold cryptography. They were usually considered in small-scale in-house applications. There are recent resurge of interests of those protocols, mostly because of a diverse set of new blockchain applications, for example, cross-chain bridge [18, 49], MEV protection [50, 59, 66], censorship resistance in asynchronous consensus [42, 52], checkpointing into Bitcoin [2], and more. Those new applications raise a new fundamental challenge of deploying distributed key generation on a very large scale.

Enhancing Proof of Stake Security via All-hands Checkpointing. As one main motivational example of large-scale DKG, we elaborate on the checkpointing mechanism, which aims at addressing a prominent security challenge in the Proof-of-Stake (PoS) blockchain known as *long-range attacks* [62]. At a high level, in a PoS blockchain, validators (with stakes) are in charge of proposing blocks; as time evolves, validators' secret keys could be leaked or simply sold (when all coins are spent) to the adversary, who can now easily create a fork from a historic block using the corresponding secret keys. For a newly joined node, such a fork is also considered valid. This long-range attack hints at an inherent vulnerability of "revisionist history" in PoS (and other resources such as space) blockchain.

One promising defense approach is to leverage proof-of-work (PoW) blockchains which are immune to such attacks, particularly to periodically let *the whole PoS network* (e.g., all validators) produce checkpoints and put them into Bitcoin. A recent work, Babylon [63], let all validators generate checkpoints via multi-signatures and select some of them to post the checkpoints to Bitcoin. It follows that the number of Bitcoin transactions needed for each checkpoint grows linearly in the number of PoS validators (to at least put a bit vector for indicating corresponding public keys). Particularly, for a PoS chain with 2^{12} validators (e.g., Filecoin), the annual cost would be over 6 million USD (using the Bitcoin price retrieved on March 31st, 2024, and assuming checkpoints are created hourly).

Instead, Filecoin proposed a blueprint for the checkpointing mechanism called Pikachu [2] via threshold Schnorr signature [35, 48]. Specifically, all validators of a PoS chain need to run a DKG protocol for *every epoch*, and the resulting public keys will serve as Bitcoin addresses. At epoch i, the validators from epoch $i-1$ will jointly create a Bitcoin transaction via a threshold Schnorr signing protocol, which contains the state of the PoS chain at epoch $i-1$ and transfers all coins from the address created in epoch $i-1$ to the newly created address. In doing so, the Bitcoin transactions uniquely decide the state of the PoS chain in each epoch, and they are verifiably endorsed by the majority of validators. Since this approach only needs exactly one Bitcoin transaction for each checkpoint, the Bitcoin transaction fees incurred are always a small constant regardless of the scale of the PoS blockchain network and much lower than Babylon's approach.

Despite being appealing, and recent progress on threshold Schnorr signatures [8, 24, 48] could potentially be deployable, Pikachu remains in theory (or toy prototype) as all validators have to jointly run a DKG protocol for every checkpoint (as validator set evolves thus previous DKG cannot be reused). Particularly, even a moderate-scale PoS chain can have thousands of validators. Moreover, some applications need to be done in a "timely" manner, which makes the task more challenging. For example, Filecoin currently has around 2^{12} validators, with anticipated growth to 2^{14} validators in the future, while checkpoints may be supposed to be created hourly (as suggested in [63]). To deploy the Pikachu checkpointing into Bitcoin mechanism [2] for real-world blockchains (for example, Filecoin), we need to design a scalable DKG protocol that can be efficiently run among all validators in popular public blockchains.

Existing DKGs are practically infeasible at a whole-chain scale. Let us first briefly introduce the common paradigm for DKG protocols, which subsumes most DKG constructions, including [35, 46, 56, 65, 70], to illustrate the astronomical communication and computation costs of existing DKGs in a large scale.

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 (as a dealer of a verifiable secret sharing (VSS) [56]) $sk_j^{(i)} = f_i(j)$ to other P_j and *broadcast* [1] a commitment, com_i, for the polynomial $f_i(X)$. Then, each participant P_j could verify if $sk_j^{(i)}$ is a valid share w.r.t. com_i; If there are invalid shares, the participants will collectively engage in a *complaint* phase, where they *broadcast* complaints and identify the set of qualified dealers $\mathsf{Qual} \subset [n]$, ensuring that all transmitted secret shares are valid. The final secret share for P_i is $sk_i = \sum_{j\in\mathsf{Qual}} sk_i^{(j)}$, and the aggregate secret key is $sk = \sum_{j\in\mathsf{Qual}} sk^{(j)}$.

The DKG scheme by Kate, Zaverucha, and Goldberg [46] (and its recently improved version by Zhang et al. [70], dubbed KZG hereafter) represents the state of the art following this paradigm. In KZG, the commitment to the polynomial f_i is constant in size and enables validating a secret share with constant-sized information. In an optimistic case, when all participants are honest, KZG protocol can remain efficient even for large-scale deployment, as demonstrated in [70]. However, since there is a constant fraction of adversarial participants, its performance got dramatically worse. Particularly, when another node complains about a dealer, the dealer shall *broadcast* the corresponding secret share for public verification. Hence, in facing $O(n)$ malicious nodes, each complaining $O(n)$ nodes, there are $O(n^2)$ shares to be broadcasted, and every node shall verify these shares. Indeed, improving the adversarial case performance is the major open problem left by [65].

We can readily anticipate that both communication and computation costs would skyrocket as the scale increases, as seen in scenarios like Filecoin validators. For instance, with 2^{12} participants in the DKG protocol, the entire network would need to transmit *tens of terabytes* of data, while each node would have to allocate *multiple hours* to verify shares (in response to complaints) to produce just one public key![2]

We remark that publicly verifiable secret sharing (PVSS) can eliminate the complaint phase in DKG [5, 17, 32, 38], as each party could now broadcast all "encrypted" shares and enable the public verification immediately. However, existing PVSS-based DKGs are either even more costly than KZG [32, 41] or only generate group-element secrets, while mainstream threshold cryptographic protocols like threshold Schnorr signatures use field-element secrets. In addition, no existing PVSS schemes with field-element secrets are provably secure against adaptive attackers [5], while adaptive security is desired in practice[3].

Besides those high costs, one extra challenge may make things even worse: in PoS chains, validators usually have different *weights* (proportional to the number of held stakes), while a threshold of weights is assumed to belong to honest validators. Simply viewing a validator as a participant in DKG can be leveraged by an adversary to amplify its power: the adversary can choose to corrupt many validators with small weights and eventually control the majority of DKG participants within its budget. A naive approach is to allocate different numbers of sub-IDs proportional to their weights. Each sub-ID is then treated as an independent participant. 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[4].

It follows that the following question remains:

Could we have a practically feasible DKG protocol that enables all-hands participation in blockchains with weighted validators, as well as being adaptively secure?

1.1 Our Results

In this article, we give an affirmative answer to this question. We proceed in two main steps:

A scalable and adaptively secure DKG. Our primary result is a practical DKG protocol (called Any-Trust DKG) for DLog-based cryptographic systems. We compare our scheme with state-of-the-art efficient DKG constructions[5] in Table 1 and discuss more related works in Sect.10. Particularly, our DKG protocol features:

– *Efficiency:* It enjoys (quasi-)linear (in n) per-node computation complexity[6], even in adversarial cases. Additionally, the size of *all* data to be broadcasted also only grows linearly in n. In contrast, previous constructions suffer from quadratic per-node computation and quadratic broadcast overhead. Specifically, as our experiments will demonstrate, for $n = 2^{12}$, each node can complete all computation tasks in approximately 27 seconds, with the data to be broadcasted totaling around 7.5 MB in size.

– *Security:* Our protocol achieves optimal resilience and satisfies the security definitions achievable by classical DKGs. Specifically, in the presence of adaptive adversaries (who cannot retract messages

[1] Broadcast satisfies *agreement*, i.e., all parties receive the same message even when the sender is malicious. Thus, it is more complicated and expensive than multicast.

[2] For detailed numerical estimates and comparisons, we refer to Sect.9.2.

[3] While there are two very recent PVSS-based DKG works [3, 30] have further pushed down the asymptotical complexity of DKG, they are either only with group-element secrets or statically secure.

[4] https://filfox.info/en/ranks/power. It has a total mining power of around 25 EB, while the power unit is 32KB, so 674 trillion sub-IDs are needed.

[5] We focus on fully synchronous networks; asynchronous or partial synchronous DKGs [1, 25, 33] are not included in the table, as their implementations cannot simply leverage a broadcast channel and appear to be more expensive.

[6] Although evaluating $O(n)$-degree polynomials at $O(n)$ points inherently causes $O(n \log n)$ computation, we only require $O(n)$ expensive group exponentiations.

Table 1: Comparison with the state-of-the-art DKGs for DLog-based Cryptography.

Schemes	Resilience	Adap.?*	Comm. Cost (total)*** Good†	Comm. Cost (total)*** Bad†	Comp. Cost (per node)** Good†	Comp. Cost (per node)** Bad†
Pedersen [56]	1/2	✓	$O(n\mathcal{B}(n\lambda)) + O(n^2\lambda)$	-	$O(n^2)$	-
KZG[46, 70]	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 [37]‡	1/2	✗	$O(n\mathcal{B}(n\lambda))$		$O(n^2)$	
GJM+ [43]‡	$\log n/n$	✗	$O(n\mathcal{B}(\lambda) + \log n\mathcal{B}(n\lambda))$		$O(n\log^2 n)$	
BHK+[10] §	$\approx 1/4$	✓	$O(C\mathcal{B}(C\lambda)) + O(C\mathcal{M}(C^2\lambda))$	-	$O(C^3)$	-
Ours (Sect.5)§	1/2	✓	$O(s\mathcal{B}(n\lambda))$	$+O(n\mathcal{M}(s\lambda))$	$O(sn)$	-

*Adap.? asks if the protocol is adaptively secure, and we accept the relaxed definition from [4]

**Comp.Cost measures the number of group exponentiation operations performed by each node.

*** Comm.Cost measures the total communication cost, where $\mathcal{B}(\ell)$ (or $\mathcal{M}(\ell)$) denotes the cost of one node broadcasting (or multicasting) ℓ bits to the network, and $O(\ell)$ means there are $O(\ell)$ bits sent by honest nodes over pair-wise channels.

†For both communication and computation, Good 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.

§BHK+ and ours are committee-based approaches. For ensuring the quality of the committee with high probability (say $1 - 5 \times 10^{-9}$, as adopted by Algorand [21]), BHK+ needs the committee size of $C \approx 6000$ (estimated based on [9]), while ours only needs $s = 38$ (further analysis available in Table.2).

sent by honest parties, as in many settings, such as Algorand [39]), our scheme complies with the oracle-aided algebraic simulatability, which was recently introduced in [4] for capturing the adaptive security of many practical DKGs including Pedersen [56] and KZG.

We remark that our primary goal is to enable massive-scale DKG; we may use resources that are naturally available in the application settings. Compared with the classical DKG schemes, our protocol additionally leverages one common coin that is generated after all participants' public keys are determined (as the YOSO-model DKG [10]), which is available in most blockchains. Nonetheless, it enables a distributed randomness beacon [22] for continuous coin generation, by applying threshold unique signatures with secret shares from the DKG [15] . We introduce a set of techniques for the efficient, adaptively secure DKG (detailed in the next section).

A generic transformer for weighted distributed protocols. We present a generic sub-ID allocation mechanism that enables us to efficiently apply conventional distributed protocols in the weighted setting. Our sub-ID allocation mechanism deterministically decides the number of sub-IDs for each validator according to the weight distribution, such that every sub-ID will be viewed as an individual participant in the subsequent protocol.

The trivial sub-ID allocation method that precisely preserves the portion of each validator's weight may need to issue tremendously many sub-IDs. In contrast, 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 system components) and the honest ratio needed in threshold cryptography (usually just above 1/2). Particularly, our sub-ID allocation method is lossy-yet-qualified, guaranteeing that more than half of the sub-IDs will be issued to honest participants if they possess over 2/3 of the weights, and therefore it can be much more *compact*. The number of sub-IDs issued by our method for n validators is probably at most $2n$, regardless of the weight distribution. For the real-world weight distributions of blockchain validators, the issued sub-IDs can even be fewer. For example, our method gives only 1688 sub-IDs instead of 674 trillion to the 3700 Filecoin validators. Compared with concurrent work in [27], ours issues fewer sub-IDs for large validator sets like Filecoin's. More details can be found in Sect.6.

Implementation and Evaluation. We implement our protocol in Java and deploy it on AWS EC2 instances with 16, 32, 64, 128, and 256 nodes. The results demonstrate that our protocol scales effectively and completes within a few seconds (adding some ledger waiting time, which could vary depending on the blockchain if we instantiate the broadcast channel via a distributed ledger.). Additionally, we conduct computational time tests for various values of n, ranging from 2^9 to 2^{15}. In comparison with the state-of-the-art DKG protocol KZG [46], 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, while KZG's cost in the adversarial case experiences a dramatic increase. Notably, for $n = 2^{12} \sim 2^{14}$, a node in our protocol can finish all computation tasks within around 26 ~181 seconds, even facing the maximal amount of Byzantine nodes. The total amount of data to be broadcasted is around 7.5 ~29.8 MB; If the nodes broadcast the data by posting it on the Filecoin blockchain, it takes around 5 minutes ~20 minutes. The experimental results show that our protocols effectively enable massive-scale DKG deployment.

Deployment friendliness: It is worth noting our DKG is more friendly for large-scale deployment since our DKG makes exclusive use of multicast channels (besides broadcast channels), which can be efficiently implemented, e.g., with gossip protocols and does not require a node to know the IP addresses of all other peers. In contrast, both Pedersen DKG and KZG DKG require pair-wise *private* channels for their efficiency claims. While it is not infeasible for large-scale deployment, it does add extra difficulties and overheads, particularly in public blockchain settings.

Application: better all-hands checkpointing. We then apply our techniques to realize the checkpointing blueprint Pikachu of Filecoin [2] that requires all validators to participate. After our optimized sub-ID allocation, we execute a DKG and a threshold Schnorr signature [61] among these 1688 sub-IDs to create a checkpoint. With our Any-Trust DKG, the DKG phase only incurs around 3MB 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 [35]

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 [48].

Compared with the existing checkpointing scheme Babylon [63], in which the number of Bitcoin transactions per checkpoint grows linearly to the scale of the blockchain, ours/Pikachu only requires exactly *one* Bitcoin transaction for each checkpoint. This difference is reflected in the monetary cost. As an example with Filecoin, the estimated Bitcoin transaction fee incurred annually using Babylon would be over six million USD, while only 26,048.8 USD using ours/Pikachu. More details can be found in Sect.8.

2 TECHNIQUE OVERVIEW

We give a high-level overview of how we leverage various techniques to lead to our DKG protocol. Through the analysis of how DKG usually works, we observe one major reason for the inefficiency (both high communication and high computation costs) is due to the following simple facts: everyone *broadcast* shares to everyone, and broadcast channels are expensive!

Starting observation for efficiency: Selecting an any-trust group as VSS dealers. Our starting observation is that letting all participants act as dealers of verifiable secret sharing (VSS) schemes is actually *unnecessary*. 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 P_j, and $\mathsf{Qual} \subset [n]$ is the set of all qualified dealers. However, the existence of **one** honest dealer would suffice for both secrecy and robustness. Particularly for secrecy, a uniformly sampled secret $s^{(j)}$ contributed by an honest P_j could conceal sk to the adversary who may corrupt all other dealers. For robustness, even when the other dealers behave arbitrarily (e.g., go offline), one honest dealer ensures the set Qual is non-empty, and thus sk is well-defined.

Therefore, we propose utilizing a small group of representatives, called an "any-trust" group (as introduced in the context of anonymous communication [68]), where we trust at least one member of the group is honest but do not need to know which one to trust. Note that such an any-trust group can be obtained by randomly sampling from the whole population (with an honest majority). Notably, the size s of an any-trust group can be as small as a few tens in practice, which is in stark contrast to that of a group with an honest majority, which can be up to thousands.

If focusing on *static* adversaries who cannot corrupt parties during the protocol execution, the observation alone already leads to an efficient solution. Particularly, before the complaint phase, there are only s commitments in total to broadcast and only s secret shares for each party to verify. In the worst case, each party at most needs to verify $O(sn)$ shares, which is still feasible.

Challenges and techniques for adaptive security. Achieving efficiency while preserving adaptive security is, however, non-trivial as the adversary does have the budget to corrupt the *entire* any-trust group. Indeed, there are multiple difficulties from different layers, and we need different techniques to conquer them.

Preventing the damage of corrupting the entire any-trust group. We adopt the standard techniques from existing adaptively secure Byzantine agreement protocols [26, 39] to prevent the damage of entire any-trust group corruption. Particularly, we use the following techniques or assumptions.

VRF-based sortition. We use the verifiable random function (VRF) based sortition [39] to select the any-trust group, such that only a party itself knows whether it has been selected, which prevents an adaptive adversary from targeting the group before the group members send out their first messages.

Memory erasure (assumption). Once the any-trust group of dealers sends out messages, the adversary will be aware of their identities and proceed to corrupt them. We, therefore, require all these dealers to *erase* all internal states related to dealing secrets (but not the long-term secret keys for signing and decryption) at the same time they send messages. Consequently, even when the adversary corrupts them, it cannot learn the secrets dealt by them.

Forward-secure signatures. However, the adversary can still violate the robustness and secrecy by sending different messages on behalf of newly corrupted dealers. In this case, an initially honest dealer may be disqualified by the network due to the disturbing messages sent by the adversary. To prevent such an attack, we apply forward-secure signatures [26], which ensures no further valid messages can be generated in this round after the dealer erases its secret states.

Efficiently deciding the qualified dealer set with silent dealers. Recall that in the conventional DKG schemes, including KZG, a dealer shall repudiate complaints (that he was silent) by broadcasting the corresponding shares for public verification. However, in our construction, all dealers may be corrupted after the dealing phase. Also, for security, they have already erased all internal states and cannot repudiate anyway. We must enable the network to decide on the qualified set of dealers while the dealers remain silent.

A seemingly immediate solution is to use a non-interactive publicly verifiable secret sharing (PVSS) scheme, such that the qualified set can be determined without the complaint phase. However, as we discussed before, existing PVSS schemes that produce field-element secrets are not adaptively secure. Moreover, non-interactive PVSS schemes [36] involve computationally expensive non-interactive zero-knowledge (NIZK) proofs for the validity of encrypted shares, which makes the DKG protocol suffer from high computation costs.

Instead, we tackle the problem by designing *publicly verifiable complaints*, such that a dealer can be disqualified immediately once such a complaint against it has been presented, without the need for further repudiation from the dealer. There are two types of complaints: (1) the dealer does not send anything to the receiver. (2) the dealer sent an invalid share to the receiver. To make type (1) public verifiable, we let each dealer in the deal phase *broadcast* the vector of *all encrypted* shares under the receivers' public keys, such that everyone can check the existence of ciphertexts. [7] For type (2), we leverage *verifiable decryption*: if the decrypted share is invalid, the receiver can generate a NIZK proof showing the share is the correct decryption, which, together with the share itself, serves as a publicly verifiable complaint.

We stress that while our design and conventional PVSS schemes share some similarity in that each dealer *broadcast all encrypted shares*, ours does *not* require the dealer to prove the validity of

[7] We remark that one can broadcast the vector of shares at a marginal cost increase compared to broadcasting one share by leveraging the effective broadcast extension trick, such as [54] and ours in Sect.7.

encrypted shares (which is significantly more costly on computation). First of all, it avoids significant computation costs while still enabling a very efficient complaint phase. Moreover, as we will elaborate in the following, this paradigm shift enables us to circumvent the adaptive security barrier in PVSS schemes.

Simulating encrypted shares in the face of adaptive corruption. Now an honest (selected) dealer needs to broadcast the sequence of encrypted shares $(\mathsf{Enc}(ek_1, f(1)), \mathsf{Enc}(ek_2, f(2)), \ldots, \mathsf{Enc}(ek_n, f(n)))$, where ek_i is the public encryption key of the party P_i, and f is the secret polynomial such that $f(0)$ defines his secret. When an adversary corrupts P_i, it knows the decryption key dk_i and thus the decrypted share $f(i)$. However, in the security proof, a simulator should not know $f(0)$ and all $f(i)$'s, while it needs to generate all ciphertexts to simulate an honest dealer. Under adaptive corruptions, we essentially need a non-committing public key encryption scheme [14], which enables the simulator to generate valid ciphertexts without knowing the plaintexts and later open the ciphertext to an arbitrary value. However, general non-committing encryption is impossible in the standard model [55]. We may employ a public key encryption scheme in the random oracle model to circumvent this difficulty [8]. Particularly, let us think about the hybrid ElGamal encryption: we have $ek = g^x \in \mathbb{G}$, $dk = x \in \mathbb{Z}_p$, and the ciphertext c in the form of $(g^r, \mathsf{Hash}(ek^r) \oplus m)$, where $g \in \mathbb{G}$ is the generator of the group $\mathbb{G}$ of prime order p, $r \in \mathbb{Z}_p$ is the fresh encryption randomness, m is the plaintext, Hash is a hash function modeled as a random oracle, and $\oplus$ is the XOR operation on the message space (assuming binary encoded for simplicity). Then, the simulator could first generate a ciphertext as (g^r, u), where u is uniformly sampled from the plaintext space. Later, when the plaintext m is known, the simulator programs the random oracle such that $\mathsf{Hash}(ek^r) = u \oplus m$, which opens the ciphertext to m.

Preventing leakages due to publicly verifiable complaints. We observe that our publicly verifiable complaints expose the decrypted results to the public, which, in some sense, provides a *decryption oracle* and can potentially be leveraged by malicious nodes to break the confidentiality of the encryption scheme. We can patch this issue by employing a chose-ciphertext-attack (CCA) secure encryption scheme. However, as we already have many other requirements for the encryption scheme, we must be careful to ensure all requirements are compatible. For example, the encryption scheme must be non-committing and require programmable random oracles, which cannot coexist with standard CCA approaches like Naor-Yung [53]. Meanwhile, our complaint phase needs efficient proof of decryption, which means the ciphertext must preserve some structures to enable efficient proof systems.

We use a signature of knowledge [19] to handle this issue. Specifically, for the hybrid ElGamal encryption whose ciphertexts are in the form of $(c_0 = g^r, c_1 = \mathsf{Hash}(ek^r) \oplus m)$, we require the dealer P_i who produces (c_0, c_1) to sign its ID i using the knowledge of r against $c_0 = g^r$. Then, in the security proof, the simulator could *extract* r from the signature of knowledge, which enables the simulator to know the encrypted share without the help of a decryption oracle. We will see other benefits of this approach when we detail the concrete encryption scheme.

[8] Now we can see the choice that we do not prove the validity of encrypted shares is critical, as otherwise, we may not use random-oracle model PKE.

Further optimizations to DKG: After overcoming the difficulties of adaptive security, we turn back to optimizing the performance.

Reducing communication of complaints by any-trust group again. A straightforward complaint phase is to let all nodes directly broadcast their (verifiable) complaints to the network. In practice, it means there could be $O(n)$ broadcast again, which can incur an unpleasant overhead. In addition, the cost cannot be reduced by our extended broadcast channel techniques either. Our broadcast technique enables one node to broadcast large-size messages, but now there are many senders.

We optimize the complaint phase via the following observation: one valid complaint is enough to disqualify a dealer, and thus, there is no need to include all complaints in the broadcast channel. We, therefore, design a complaint phase with the following three steps. First, each node disseminates the complaints using a multicast channel so that all nodes receive all complaints made by all other honest nodes. Second, we sample an any-trust group again, and let the group members deduplicate the complaints. Each group member will maintain a concise complaint list that contains at most one complaint for each dealer and all dealers complained by honest nodes. Finally, we let the group members broadcast their complaint lists, which guarantees that all malicious dealers will be disqualified. With the optimized complaint phase, there are $O(s)$ any-trust group members, each posting at most $O(s)$ complaints, where s is the size of an any-trust group.

On the choice of VSS/polynomial commitments. While the VSS scheme in KZG DKG [46] is usually believed to be the most efficient instantiation, we do not use it in our DKG scheme due to the following considerations: (1) The polynomial commitment scheme in KZG VSS necessitates a structural common reference string, and securely establishing it in decentralized applications requires additional efforts. (2) The communication benefits of the VSS scheme do not exist in our setting. Although its commitment size is constant, we need to broadcast all encrypted shares (and their encrypted proofs) anyway. (3) The generated public key is in a pairing-friendly group. We need to make an extra effort to adapt it for Schnorr signatures.

Instead, we employ a VSS scheme based on a more classical polynomial commitment. Specifically, the commitment to a t-degree polynomial f is the form of $g^{f(0)}, g^{f(1)}, \ldots, g^{f(n)}$, where $n > t$ is the number shares needed to distribute. By the checking technique from Scrape [17], a receiver could verify the $n+1$ group elements committing to a t-dgree polynomial at the cost of $O(n)$ group operations. Then, to verify each share $f(i)$, one just needs to perform one group exponentiation operation, such that verifying $O(n)$ shares from the dealer just costs $O(n)$ group operations, which guarantees a computationally efficient complaint phase.

Using multi-recipient encryption. For the PKE scheme, we have proposed the hybrid version of ElGamal, which is *non-committing* and supports verifiable decryption. In our DKG, we use the multi-recipient variant of it [7], which reuses the g^r component across ciphertexts under different public keys. It greatly reduces the broadcast cost, making the ratio of ciphertext size and share size close to 1. Moreover, recall that we use proof of knowledge of r to prevent leakages from decryption oracles, and using multi-recipient encryption will only incur one proof of knowledge by each dealer.

Optimization to broadcast channels: A practical extension trick. Two primary approaches for broadcast channels include using Byzantine broadcast (BB) protocols [21, 28, 39] or utilizing existing infrastructure like blockchains. Implementing a large-scale BB protocol can be intricate and susceptible to errors; thus, using established blockchains is an attractive, simpler, and modular alternative. However, on-chain storage is generally an expensive and scarce resource. While the broadcast cost in our DKG for thousands of participants has been reduced to a few Megabytes, it can still be a considerable burden for blockchains.

Therefore, we present a practical extension to a blockchain-based broadcast channel by leveraging a multicast channel and a data dispersal network (DNN) like IPFS [67]. Our design is simple and modular, retaining the major benefits of using blockchain, and it enables a sender to broadcast an *arbitrarily long* message while incurring *constant* on-chain storage cost. Though it may be folklore to write digests alone into a blockchain to save bandwidth, we are unaware of any design with a formal agreement guarantee. We believe this component may be of independent interest. More details are in Sect.7

3 MODEL AND GOAL

Communication model. We assume the network is synchronous, and protocols proceed by rounds. Every participant has access to multicast and broadcast channels with different guaranteed delivery time. They both achieve *validity*, while broadcast channel additionally guarantees *agreement*.

Validity. When an honest node sends a message via this channel, all honest nodes can receive this message by the end of the 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 behaviors. Particularly, the adversary controls what messages a corrupted node will send in the same round it gets corrupted. However, messages already multicasted or broadcasted by node i before i become corrupted *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 $f(n) \leq \mathrm{negl}(n)$, if for every positive integer c, there exists an n_0 such that for all $n > n_0$, $f(n) < n^{-c}$. For a set $\mathbb{X}$, the notation $x \leftarrow\$ \mathbb{X}$ signifies sampling x uniformly from $\mathbb{X}$. We use $y \leftarrow A(x_1, x_2, \cdots)$ to represent running A with inputs $x_1, x_2, \cdots$ and uniform randomness and outputting y. 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 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 $sk_i \in \mathbb{Z}_p$. This setup allows any subset of $t+1$ honest parties to reconstruct the secret key sk of pk.

We follow the *oracle-aided algebraic simulatability*, which was recently proposed by Bacho and Loss [4] for capturing the adaptive security of many practical DKG schemes. This definition focuses on *algebraic* adversaries.

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 although 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 [4].

Additionally, we consider "key-expressibility", as introduced in [43], against static attackers. This property is suitable for instantiating the key generation of re-keyable primitives like BLS and ElGamal. It also works with Schnorr signature as recently shown in [43, 61]. Formal definitions are provided in Appendix **??**.

Definition 3 (Key-expressability [43])**.** 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 $pk = pk'^{\alpha} \cdot pk_1$.

4 PRELIMINARIES

Verifiable Random Function. 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 take the DDH-based VRF scheme from [40] as our instantiation. A VRF scheme consists of three algorithms: (1) $\mathsf{KeyGen}(1^\lambda)$ generates a verification key vk and the secret evaluation key sk; (2) $\mathsf{Eval}(vk, sk, x)$ evaluates the function with sk on the input x, and outputs y along with a proof π. (3) $\mathsf{Verify}(vk, x, y, \pi)$ verifies if y is the honest evaluation output on x with the secret key of vk.

A secure VRF satisfies (1)*Pseudorandomness*: the function values are pseudorandom, even given the public key and the proofs; (2) *Completeness*: it always holds that $\mathsf{Verify}(vk, x, \mathsf{Eval}(vk, sk, x)) = 1$; and (3) *Uniqueness*: it is infeasible to generate a public key vk, an input x, and two different (y_1, π_1) and (y_2, π_2), such that

$$\mathsf{Verify}(vk, x, y_1, \pi_1) = \mathsf{Verify}(vk, x, y_2, \pi_2) = 1.$$

(4) *Unpredictability under malicious key generation*: if the input x has enough entropy (i.e., cannot be predicted), then the correct output y is indistinguishable from a uniformly random value, no matter how the VRF keys are generated. Formal definitions can be found in [6, 51].

VRF-based sortition. We introduce the standard VRF-based sortition scheme below and will use it as a black box in our DKG protocol. (1) $\mathsf{Setup}(1^\lambda)$. Each user generates their VRF key pair (vk, sk) and publishes vk. A public randomness rand is sampled independent of the key generation. (2) $\mathsf{Sortition}(vk, sk, \mathsf{rand}, \mathsf{event}, \mathsf{ratio})$. 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 $\frac{y}{\mathsf{max}} \leq \mathsf{ratio}$. If failed, abort. Otherwise, return (y, π) as the credential of being selected. (3) $\mathsf{Vrfy}(vk, \mathsf{rand}, \mathsf{ratio}, \mathsf{event}, \mathsf{credential})$. It verifies the credential by validating the VRF output y and checking if $\frac{y}{\mathsf{max}} \leq \mathsf{ratio}$.

In above, 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.

Security of VRF-based sortition. It is easy to argue when the underlying VRF satisfies the security properties defined above, and rand is sampled independently of the key generation, the VRF-based sortition outcome is computationally indistinguishable from the outcome of a process where each user is elected with an independent probability of ratio. Our further analysis is based on this fact.

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. The formal security definitions and secure instantiations are available in [44].

Multi-recipient encryption. We use the multi-recipient hybrid ElGamal encryption. Let g be a generator of $\mathbb{G}$, and let Hash be a hash function whose output space is the message space (which we assume is binary encoded and $\oplus$ is the XOR operation). The encryption scheme can be described as follows: (1) $\mathsf{Gen}(1^\lambda)$ outputs $(ek = g^x, dk = x)$, where $x \leftarrow\!\!\$\ \mathbb{Z}_p$. (2) $\mathsf{MREnc}(ek_1, \ldots, ek_n, m_1, \ldots, m_n)$ outputs the ciphertexts $(c_0, c_1, \ldots, c_n)$, where $c_0 = g^r$ for some uniformly sampled $r \in \mathbb{Z}_p$, $c_i = \mathsf{Hash}(ek_i^r) \oplus m_i$ for $i \in [n]$. (3) $\mathsf{Dec}(ek_i, dk_i, c_0, c_i)$ outputs $m = \mathsf{Hash}(c_0^{dk_i}) \oplus c_i$.

We use the above algorithms in our DKG construction, but we directly reduce our DKG to the underlying DDH assumption without going through the security abstraction of the encryption scheme. This is because the security properties we need from the encryption are non-standard (as we discussed in the technique overview), and we would like to avoid further distractions.

Signature of Knowledge. A signature of knowledge (SoK) scheme is defined w.r.t. an NP relation R. It can be either in the common reference string model or in the random oracle model. We focus on the random-oracle-model instantiations and thus omit the algorithm for generating the common reference string. A user with the witness x of a public statement y such that $(y, x) \in R$ can sign any message m via the signer algorithm $\mathsf{SoK.Sign}(y, x, m) \to \sigma$. Later, another user can verify if m was signed by someone with the knowledge of the witness w.r.t. y via the verifier algorithm.

A secure SoK scheme satisfies the following properties: (1) *Simulatability*: there is an efficient simulator algorithm that can produce a valid signature under any statement y without using the witness x, and the produced signatures are indistinguishable from honestly generated signatures. (2) *Extractability*: There is an efficient extractor algorithm that can extract the witness x from a valid signature produced by the adversary under a statement y, even when the adversary has seen some simulated signatures. In the random oracle model, both the simulator and extractor are allowed to program the random oracle. Formal definitions are available in [19].

In this paper, we use an SoK in the random oracle model for the DLog relation, *i.e.*, for $y \in \mathbb{G}$ and $x \in \mathbb{Z}_p$, we have $(y, x) \in R$ iff $y = g^x$. Note that such an SoK is well-studied and can be instantiated with Schnorr signature scheme.

NIZK. The proof of decryption used in our DKG is a NIZK proof system for the decryption correctness. In the random oracle model, a NIZK for an NP relation R consists of a prover algorithm Prove, which on inputs a statement y and its witness x outputs a proof π,

and a verifier algorithm Vrfy which validates the proof against the statement y.

In this work, we require the NIZK to satisfy the following properties: (1) *Completeness*: For every $(y, x) \in R$, it holds that $\mathsf{Vrfy}(y, \mathsf{Prove}(y, x)) = 1$. (2) *Zero-knowledgeness*: There exists an efficient simulator algorithm that can produce a valid proof for any statement y without knowing x, and the simulated proofs are indistinguishable from honestly generated proofs. (3) *Simulation soundness*. For a statement y, if there is no witness x s.t. $(y, x) \in R$, then it is infeasible for an efficient adversary to produce a valid proof for y, even when the adversary has seen simulated proofs. Formal definitions are available in [60].

We use a NIZK for proof of decryption correctness w.r.t. the encryption scheme. It consists of a prover algorithm, PKE.Prove, which on inputs $(c_0, c_i, ek_i, dk_i, m)$ produces a proof Γ, and a verifier algorithm $\mathsf{PKE.Vrfy}(c_0, c_i, ek_i, m, \Gamma)$ which checks whether m is the correct decryption from (c_0, c_i). Particularly, $\Gamma = (m, c_0^{dk_i}, \pi)$. 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) [20].

Scrape's polynomial commitment. We use the polynomial commitment scheme from Scrape [17]. Particularly, let g be the generator of $\mathbb{G}$ of prime order p, let f be a t-degree polynomial over $\mathbb{Z}_p$, and let $n > t$ be an integer. Then, the commitment to the polynomial f is

$$(\mathsf{cm}_0 = g^{f(0)}, \mathsf{cm}_1 = g^{f(1)}, \ldots, \mathsf{cm}_n = g^{f(n)}).$$

One can check whether these group elements commit to a t-degree polynomial by performing the following steps: (1) Sample an $(n-t)$ -degree polynomial $q(X) \in \mathbb{Z}_p[x]$, and compute the dual code: $\mathsf{cm}_\tau^\perp = \frac{q(\tau)}{\prod_{j=0, j\neq\tau}^{n}(\tau - j)}, \forall \tau \in [0, n]$. (2) Check whether $\prod_{\tau=0}^{n}(\mathsf{cm}_\tau)^{\mathsf{cm}_\tau^\perp} = \mathbf{1}_\mathbb{G}$, where $\mathbf{1}_\mathbb{G}$ is the identity element of $\mathbb{G}$.

It is worth noting that the first step can be reused to check different commitments. The effectiveness of the checking process is determined by the following lemma.

Lemma 1 ([17]). *For any* $(\mathsf{cm}_0, \mathsf{cm}_1, \ldots, \mathsf{cm}_n) \in \mathbb{G}^{n+1}$, *if*

$$\prod_{\tau=0}^{n}(\mathsf{cm}_\tau)^{\mathsf{cm}_\tau^\perp} = \mathbf{1}_\mathbb{G},$$

then with an overwhelming probability there exists a t-degree polynomial f, *such that* $cm_i = g^{f(i)}$ *for* $i \in [0, n]$.

After that, a share $f(i)$ can be validated by checking whether $g^{f(i)}$ equals cm_i.

5 OUR DKG PROTOCOL

Following the technique overview in Sect.2, we present our Any-Trust DKG in Sect.5.1 based on the building blocks in Sect.4, and analyze it in Sect.5.2.

5.1 The Construction

The construction is based on building blocks such as PKE (along with the proof decryption system), the forward-secure signature FS, the VRF-based sortition VRF, and SoK.

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, determine the group $\mathbb{G}$ of prime-order p and its generator g.

PKI Setup: Every participant 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.

The setup also determines 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 underlying channels are established during this setup phase.

Protocol Details. Post-setup, all participants collaboratively run our DKG protocol, detailed in Fig.1. 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 as follows:

–**Round 1.** Nodes initially determine if they're selected as dealers. If not, they refresh the secret key and exit the round (lines 1-4). Elected dealers sample a t-degree polynomial f to decide secret shares $sk_i = f(i)$, commit to $sk_0, \ldots, sk_n$, encrypt shares $sk_1, \ldots, sk_n$ using others' encryption keys (lines 6-8), and sign their ID using the knowledge of r w.r.t. $c_0 = g^r$ in the ciphertext (line 9). Dealers then sign the commitments and ciphertexts, update their signing keys, erase secret information, and broadcast the signed commitments and ciphertexts (lines 10-14).

–**Round 2.** Nodes receive the broadcasted messages (line 1). For every message, they authenticate the signature (line 8); if failed, they move to the next message. Otherwise, they validate its format, the VRF sortition certificate (lines 10-11), and the signature of knowledge (line 13). Moreover, they ascertain if the committed values match valid coefficients of a t-degree polynomial (lines 3-5 and 12-13). If a transcript fails verification, the dealer is instantly disqualified (line 14). Otherwise, they check the decrypted share's validity against the commitments and, if inconsistent, generate a verifiable complaint against the dealer (lines 15-17). 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 6-17). They then sign and broadcast this complaint list (lines 18-21).

–**End of Round 3.** Nodes finalize the set of disqualified dealers based on received complaint lists (lines 1-13). Following that, they create the public key (shares) and secret key share by aggregating contributions from qualified dealers (lines 14-16).

5.2 The Analysis

Computation complexity analysis. We primarily focus on the computationally intensive operations, particularly the group exponentiation operation EXP, and will omit inexpensive operations like multiplication operation and hash evaluation. In Round 1, a node who is elected as a dealer needs to generate a commitment (line 6),

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

1: // determine whether it is elected as a dealer.
2: $\mathsf{VRF.Sortition}(rvk_i, rsk_i, \text{rand}, \text{“deal”}, \text{ratio}) \to \mathsf{CR}_i^{\text{deal}}$
3: **if** $\mathsf{CR}_i^{\text{deal}} = \perp$, // if not, update FS secret key and exit the round
4: **then** $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$, **erase** $\mathsf{FS}.sk_i[1]$, **exit Round 1**
5: // only elected users continue the followings.
6: sample $(a_0, a_1, \ldots, a_t) \leftarrow\$ \mathbb{Z}_p^{t+1}$, define $f(X) = \sum_{\tau=0}^{t} a_\tau X^\tau$
7: commit to the random polynomial $f(X)$: $(\mathsf{cm}_j = g^{f(j)})_{j\in[0,n]}$
8: encrypt shares: $\mathsf{PKE.MREnc}((ek_i)_{i\in[n]}, (f(i))_{i\in[n]}) \to (c_0, \ldots, c_n)$
9: sign the ID i using the knowledge of r: $\mathsf{SoK.Sign}(c_0, r, i) \to \sigma_{\mathsf{DL}}$
10: denote $\mathsf{trans}_i \leftarrow (\mathsf{CR}_i^{\text{deal}}, (\mathsf{cm}_j)_{j\in[n]}, (c_j)_{j\in[n]}, \sigma_{\mathsf{DL}})$
11: sign the transcript using FS: $\mathsf{FS.Sign}(\mathsf{FS}.sk_i[1], \mathsf{trans}_i) \to \sigma_i$
12: Updat the secret key of FS: $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$
13: **erase** $\mathsf{FS}.sk_i[1]$, $f(X)$, $(f(i))_{i\in[0,n]}$, and encryption randomness
14: **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: // prepare dual code for verifying polynomial commitments
4: sample an $(n-t)$-degree polynomial $q(X) \in \mathbb{Z}_p[x]$, compute
5: $\mathsf{cm}_\tau^\perp = \dfrac{q(\tau)}{\prod_{j=0, j\neq\tau}^{n}(\tau - j)}, \forall \tau \in [0, n]$
6: **for** $j \in \mathbb{D}$// verify each broadcast transcript as below
7: // ignore the transcript if the FS signature is invalid
8: **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 1, \sigma_j[1], \mathsf{trans}_j[1]) = 0$, **then continue**
9: // verify if it is in a good format and the sortition credential
10: **if** parse $\mathsf{trans}_j[1] = (\mathsf{CR}_j^{\text{deal}}, (\mathsf{cm}_\tau^{(j)})_{\tau\in[n]}, (c_\tau^{(j)})_{\tau\in[n]}, \sigma_{\mathsf{DL}}^{(j)})$ failed
11: $\vee\ \mathsf{VRF.Vrfy}(rvk_j, \text{rand}, \text{ratio}, \text{“deal”}, \mathsf{CR}_j^{\text{deal}}) = 0$
12: //check if $(\mathsf{cm}_\tau^{(j)})_{\tau\in[n]}$ commits to a t-degree polynomial
13: $\vee \prod_{\tau=0}^{n} (\mathsf{cm}_\tau^{(j)})^{\mathsf{cm}_\tau^\perp} \neq \mathbf{1}_{\mathbb{G}} \vee \mathsf{SoK.Vrfy}(c_0^{(j)}, \sigma_{\mathsf{DL}}^{(j)}, j) = 0$
14: **then** $\mathbb{D}_1 = \mathbb{D}_1 \cup \{j\}$// if any fails, disqualify j immediately
15: **elseif** $\mathsf{PKE.Dec}(dk_i, c_i^{(j)}) = sk_i^{(j)} \wedge g^{sk_i^{(j)}} \neq \mathsf{cm}_i^{(j)}$
16: //generate a complaint, and update the complaint list
17: **then** $\mathsf{PKE.Prove}(c_0^{(j)}, c_i^{(j)}, dk_i, sk_i^{(j)}) \to \Gamma_j$, $\mathbb{D}_2 = \mathbb{D}_2 \cup \{(j, \Gamma_j)\}$
18: //otherwise, update the candidate output list
19: **else** $\mathbb{D}_3 = \mathbb{D}_3 \cup \{j\}$, $\mathbb{C} = \mathbb{C} \cup \{(j, ((\mathsf{cm}_\tau^{(j)})_{\tau\in[0,n]}, sk_i^{(j)}))\}$
20: $\mathbb{D}_2 \to \mathsf{trans}_i[2]$, $\mathsf{FS.Sign}(\mathsf{FS}.sk_i[1], \mathsf{trans}_i[2]) \to \sigma_i[2]$
21: **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: // determine whether it is elected for broadcasting complaint list
3: $\mathsf{VRF.Sortition}(rvk_i, rsk_i, \text{rand}, \text{“agree”}, \text{ratio}) \to \mathsf{CR}_i^{\text{agree}}$
4: **if** $\mathsf{CR}_i^{\text{agree}} = \perp$, // if not, update FS secret key and exit the round
5: **then** $\mathsf{FS.Update}(\mathsf{FS}.sk_i[1]) \to \mathsf{FS}.sk_i[2]$, **exit Round 2**
6: set DisQual, CompList = $\emptyset$// start to deduplicate complaints
7: **for** $j \in \mathbb{R}_1$, **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 1, \sigma_j[2], \mathsf{trans}_j[2]) = 1$
8: **then for** $(k, \Gamma_k) \in \mathsf{trans}_j[2]$// check every complaint by P_j
9: // if D_k has not been disqualified, verify the complaint
10: **if** $k \notin$ DisQual, parse $\Gamma_k = (sk_j^{(k)}, \cdot)$
11: //$c_0^{(k)}, c_j^{(k)}$ are what P_i received at line 1 of round 2
12: **if** $\mathsf{PKE.Vrfy}(c_0^{(k)}, c_j^{(k)}, \Gamma_k) = 1 \wedge g^{sk_j^{(k)}} \neq \mathsf{cm}_j^{(k)}$
13: // disqualify the dealer given a valid complaint
14: **then** DisQual = DisQual $\cup \{k\}$
15: CompList = CompList $\cup \{(k, \Gamma_k)\}$
16: // stop verifying complaints from j if the complaint is invalid.
17: **else break**
18: $(\mathsf{CR}_i^{\text{agree}}, \text{CompList}) \to \mathsf{trans}_i[3]$
19: sign $\mathsf{FS.Sign}(\mathsf{FS}.sk_i[2], \mathsf{trans}_i[3]) \to \sigma_i[3]$
20: $\mathsf{FS.Update}(\mathsf{FS}.sk_i[2]) \to \mathsf{FS}.sk_i[3]$; **erase** $\mathsf{FS}.sk_i[2]$
21: **if** 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 DisQual = $\emptyset$
3: // decide the disqualified set based on broadcast message
4: **for** $j \in \mathbb{R}_2$
5: parse $\mathsf{trans}_j[3] = (\mathsf{CR}_j^{\text{agree}}, \text{CompList})$
6: **if** $\mathsf{FS.Vrfy}(\mathsf{FS}.vk_j, 2, \sigma_j[3], \mathsf{trans}_j[3]) = 1$
7: $\wedge\ \mathsf{VRF.Vrfy}(rvk_j, \text{rand}, \text{ratio}, \text{“agree”}, \mathsf{CR}_j^{\text{agree}}) = 1$
8: **then for** $(k, \Gamma_k) \in$ CompList
9: // put a newly complained dealer in the list
10: **if** $k \notin$ 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** DisQual = DisQual $\cup \{k\}$,
13: set Qual = $\mathbb{D}_3 \setminus$ DisQual
14: **output:**
15: $pk = \prod_{j\in\text{Qual}} \mathsf{cm}_0^{(j)}$, $sk_i = \sum_{j\in\text{Qual}} sk_i^{(j)}$
16: $pk_\tau = \prod_{j\in\text{Qual}} \mathsf{cm}_\tau^{(j)}$, for every $\tau \in [n]$

Figure 1: The Any-Trust DKG construction.

which takes $O(n)$EXP, encrypt shares under n different public keys (line 7), which takes $O(n)$EXP group expo. In Round 2, to verify a transcript, each node needs to validate its commitment (line 10), which takes $O(n)$EXP, and there are expected to be s transcripts to

verify. If there are the maximal amount of Byzantine nodes, an honest node may need to verify $O(n)$ VRF proofs and $O(n)$ signatures (lines 6, 8), which takes $O(n)$EXP, and generate $O(s)$ complaints, which takes $O(s)$EXP. In Round 3, each node verifies the complaints from other nodes. Note that a node P_i stops verifying complaints from P_j upon finding an invalid complaint made by P_j, and it also stops verifying complaints against a dealer once the dealer has verifiably complained. Therefore, a node verifies at most $O(n)$ complaints, which takes $O(n)$EXP. There are no group exponentiation operations in Round 4. Thus, even facing the maximal amount of Byzantine nodes, each node only needs to perform $O(sn)$ group exponentiation.

Communication complexity analysis. In Round 1, an elected node will broadcast $O(n\lambda)$-size transcript, and there are expected to be s elected node, where λ is the computational security parameter, and the sizes of a group element, a digital signature, and a VRF credential, are counted as $O(\lambda)$. So the communication cost in Round 1 is $s\mathcal{B}(n\lambda)$, where $\mathcal{B}(\ell)$ denotes the communication cost of broadcasting ℓ bits by one sender. There will be no further communication if all nodes behave honestly. Otherwise, in Round 2, each node may need to multicast $O(s)$ complaints, which in total incurs the communication complexity of $n\mathcal{M}(s\lambda)$, where $\mathcal{M}(\ell)$ denotes the communication cost of multicasting ℓ bits by one sender. In Round 3, there are $O(s)$ elected nodes each broadcasting $O(s\lambda)$-sized complaints, which incurs the communication complexity of $s\mathcal{B}(s\lambda)$. In summary, the good case communication complexity is $s\mathcal{B}(n\lambda)$, and the adversarial-case communication complexity can be $s\mathcal{B}(n\lambda) + n\mathcal{M}(s\lambda)$.

Security analysis. The t-correctness and t-consistency, which ensure all participants at the end of the protocol obtain the same public key and correct shares, follow the facts: (1) there is at least one qualified dealer, and (2) all malicious dealers will be disqualified. From the security of VRF-based sortition and the forward-secure signature, at least one honest node will be selected as a dealer, and it can successfully broadcast its valid transcript even if it later becomes corrupted, which ensures (1). From the security of the polynomial commitment, if a dealer is not complained by any node, then all honest users receive consistent shares from the dealer. Our complaint phase guarantees that every dealer who is complained by an honest node will disqualified. Therefore, we have (2).

Proving the oracle-aided algebraic simulatability is more involved. At a high level, we need to construct an efficient simulator that, on input from a sequence of group elements, produces an indistinguishable view for an adaptive adversary with the help of a DLog oracle. We follow the techniques from [4] to simulate the polynomial commitments and opening shares for corrupted nodes. However, broadcasting all encrypted shares in our protocol poses additional challenges for security proof. Particularly, the simulator needs to simulate all ciphertexts without knowing the shares. It also needs to simulate the proof of decryption without using the decryption oracle of the underlying encryption scheme. As sketched in Sect.2, we leverage the non-committing encryption and signature of knowledge to handle these challenges. Formally, we establish the following theorem.

Theorem 2. *The Any-Trust DKG satisfies t-consistency, t-correctness, and $(t, k, T_{\mathcal{A}}, T_{\mathsf{sim}})$-oracle-aided algebraic simulatability against adaptive adversaries (cf Def.2), with $n \geq 2t+1$, $k \leq n(t+1)$ 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, the NIZKPoK, the proof of decryption, 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 $j \in$ 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 P_i should have $f(i)$. If an honest P_i does not have $f(i)$, there must exist an index $j \in$ Qual such that P_i does not have $f_j(i)$. In this case, 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 that 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 sends 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.3, and the analysis for the key expressibility is in Lemma.4. □

Lemma 3. *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 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 $k = n(t+1)$, Sim can simulate each phase of Π for $\mathcal{A}$ as follows.

SETUP. Sim initializes the set of corrupted parties $C = \emptyset$, the set of honest parties $\mathcal{H} = \{P_i\}_{i\in[n]}$, 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 P_i, Sim first checks if $|C| \le t$. If the check fails, it ignores this query; otherwise, return the secret keys of P_i, and update the sets $\mathcal{H} = \mathcal{H}\backslash\{P_i\}$ and $C = C \cup \{P_i\}$.
- **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 $P_i \in \mathcal{H}$, Sim runs the *Self-Election* procedure using P_i's VRF secret key. We assume w.l.o.g. there are $s' \le n$ 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 $\mathsf{cm}_\tau^{(j)} = \prod_{\mu\in[0,t]} \zeta_{j,\mu}^{\tau^\mu}$, for every $\tau \in [0, n]$.
- Generate the ciphertext $(c_0^{(j)}, c_1^{(j)}, \ldots, c_n^{(j)})$, where $c_0^{(j)} = g^{r_j}$ for some $r_j \leftarrow\$ \mathbb{Z}_p$, and $c_\tau^{(j)} \leftarrow\$ \{0,1\}^{\lceil \log p\rceil}$ for $\tau \in [n]$.
- Use the simulated signer algorithm of SoK to sign j w.r.t. $c_0^{(j)}$ and obtain a simulated signature of knowledge $\sigma_{\mathsf{DL}}^{(j)}$.
- Broadcast $(\mathsf{CR}_j^{\mathsf{deal}}, \mathsf{cm}_0^{(j)}, \ldots, \mathsf{cm}_n^{(j)}, c_0^{(j)}, \ldots, c_n^{(j)}, \sigma_{\mathsf{DL}}^{(j)})$ along with its forward-secure signature on it.

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

- **Corruption queries.** When $\mathcal{A}$ asks to corrupt the party P_i, Sim first checks if $|C| \le t$. If the check fails, it ignores this query; otherwise, Sim queries the oracle $\mathsf{DL}_g(\cdot)$ with $\mathsf{cm}_i^{(j)}$ for all $j \in \mathcal{H}_{\mathsf{ele}}$ with its representation $(1, i^1, \ldots, i^t)$ over ζ_j. Sim will receive $\xi_i^{(j)}$ from the oracle. Then, Sim records all $\{x_i^{(j)}\}$, which are secret shares dealt by honest dealers. Sim can obtain the shares dealt by corrupted dealers for P_i by decrypting the encrypted shares using dk_i. Finally, Sim returns the secret shares for P_i and its decryption key dk_i to $\mathcal{A}$, and updates the sets $\mathcal{H} = \mathcal{H}\backslash\{P_i\}$ and $C = C \cup \{P_i\}$.
- **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 P_τ, 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** in one is in the query history. Otherwise, continue.
  When $\mathcal{A}$ queries a message x, Sim performs as follows.
  – If $x \ne \gamma_{j,\tau}$ for any j and τ, proceed as what it did in the Setup phase.
  – If $x = \gamma_{j,\tau}$ for some j and τ, checks if P_τ has been corrupted. If it has not been corrupted, then Sim first queries the oracle $\mathsf{DL}_g(\cdot)$ with $\mathsf{cm}_\tau^{(j)}$ and its representation $(\tau^0, \tau^1, \ldots, \tau^t)$ over ζ_j. Sim will receive $\xi_{j,\tau}$ from the oracle. If it is corrupted, then $\xi_{j,\tau}$ has been recorded by Sim. Finally, it sets $\mathsf{output}_{\gamma_{j,\tau}} := c_\tau^{(j)} \oplus \xi_{j,\tau}$, 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, except that whenever an honest party P_i needs to decrypt an encrypted share $(c_0, c_1, \ldots, c_n)$, Sim instead performs the following procedures for decryption.

- Use the extractor of the SoK to obtain r, such that $c_0 = g^r$. Then, use r to "decrypt" the encrypted share as $sk_i = \mathsf{Hash}(ek_i^r) \oplus c_i$.

The queries from $\mathcal{A}$ are answered in the same way as Sim did in Round 1.

Let Qual_C be the set of qualified nodes that 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 are 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. Notably, 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

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

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 $\mathcal{A}$ corrupts a party or 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 P_τ (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 $P_1, \ldots, P_t$, 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 $n(t+1)$. Specifically, Sim makes the following extra queries:

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

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 $s't + (n-s')(t+1) + s' - 1 = n(t+1) - 1$, which is still smaller than k. It is easy to verify the matrix L is invertible. □

Lemma 4. *The Any-Trust DKG satisfies the key-expressability.*

Proof. This proof is similar to the proof for Lemma.3, 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 $|\mathsf{Corr}| \le t$. 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 Lagrange interpolation in the exponent, while c_i are randomly sampled. This transcript is indistinguishable from an honestly generated one in the view of $\mathcal{A}$ and cannot be disqualified. 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 $\alpha = 1$, $sk'' = \sum sk^{(j)}$. □

Committee Size. Recall that our construction employs a VRF-based sortition to decide the committee, in which each node can be independently elected a committee member with a probability `ratio`. Assume a network of n nodes while at least h of them remains honest. Such a sortition process will produce a committee of the expected size of $s = \mathtt{ratio} \cdot n$. Then the probability p that at least one honest node being elected can be expressed as follows:

$$\mathsf{p} = 1 - (1 - \frac{s}{n})^h. \tag{1}$$

We compute the expected committee sizes necessary to ensure different values of p in networks with varying ratios of honest parties, as depicted in Table 2. For example, assuming over 51% participants of the whole network are honest, we can set the expected committee size as 38, which ensures the resulting committee contains at least one honest node with the probability of at least $1 - 5 \times 10^{-9}$. These findings are applicable to networks of any size, although for networks with $n \le 10^4$, a slightly smaller committee size may be achievable.

PR \ HR	51%	67%	80%
$1 - 5 \times 10^{-9}$	38	29	24
$1 - 2^{-30}$	41	32	26
$1 - 2^{-40}$	55	42	35

Table 2: Expected committee sizes for different probability guarantees (PR) under different honest-party ratio (HR)

6 SUB-ID ALLOCATION FOR THE WEIGHTED SETTING

In this section, we present a simple yet effective sub-ID allocation mechanism that enables us to apply a conventional distributed protocol like our Any-Trust DKG in the weighted setting. Compared with the straightforward sub-ID allocation mechanism, ours 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[9]. 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 that minimizes the number of all sub-IDs, *i.e.*, $\sum_j d_j$ is as small as possible.

[9]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.

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: $d_i = \frac{w_i}{\gcd}$. 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, we call an adjustment t-bounded for $(w_1, \ldots, w_n)$, if the adjusted values $(w'_1, \ldots, w'_n)$ satisfies $\sum_{i\in[n]} |w_i - w'_i| \le t$. Then, if $\sum_{i\in[n]} w_i \ge 3t+1$, it ensures that for any partition (A, B) over $[n]$ satisfying $\sum_{i\in A} w_i > 2 \cdot \sum_{i\in B} w_i$, it holds that $\sum_{i\in A} w'_i > \sum_{i\in B} w'_i$, given the inequality:

$$\sum_{i\in A} w'_i - \sum_{i\in B} w'_i \ge \sum_{i\in A}(w_i - \Delta_i) - \sum_{i\in B}(w_i + \Delta_i) \ge 1 \tag{2}$$

Here, $\Delta_i = |w_i - w'_i|$. Following the adjustment, 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 $f_{\gcd}(w_i) \to w'_i$ as:

$$w'_i = \begin{cases} w_i - (w_i \bmod \gcd), & \text{if } w_i \bmod \gcd < \gcd/2, \\ w_i + \gcd - (w_i \bmod \gcd), & \text{otherwise.} \end{cases} \tag{3}$$

Starting with 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.

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

Efficiency and effectiveness. Note that our AllocateSubID is only supposed to find a t-bounded $f_{\gcd}$ for its input $(w_1, \ldots, w_n)$. So we can efficiently check whether $\sum_{i\in[n]} |f_{\gcd}(w_i) - w_i| \le t$. Thus, the time-cost of AllocateSubID is $O(n \log n)$. Meanwhile, using binary search is effective since there is a general trend that the larger the gcd is, the larger adjustment is needed. It can give us a t-bounded $f_{\gcd}$ for the input with a large gcd (not necessarily optimal).

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* AllocateSubID. *It follows that $\sum_{i\in[n]} d_i \le \frac{4t+1}{\lfloor 2t/n \rfloor}$, which is around $2n$ when $n \ll t$.*

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

Table 3: Comparison with Swiper/Dora

Systems	# Parties	#Total Weights	[27]	Ours
Aptos[11]	104	8.4708×10^8	27	34
Tezos[64]	382	6.7579×10^8	75	77
Filecoin[31]	3700	2.5242×10^{19}	1895	1688
Algorand[21]	42920	9.7223×10^9	373	301

Comparison with Swiper/Dora [27]. A concurrent work, Swiper/Dora [27], 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 parties 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.

7 PRACTICAL EXTENDED BROADCAST CHANNELS

In this section, we introduce a practical extension to the blockchain-based broadcast channel. Although it is folklore knowledge that, theoretically, one may throw all messages into the ledger to facilitate a broadcast, this may incur prohibitive costs in practice, as on-chain 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 [67].

7.1 Building Blocks

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

Public Bulletin Board. We follow the model of PBB presented in [47] and extend it to support *keyword*-based retrival. A user can interact with PBB by using the following queries:

- getCounter() $\to t$. It returns the current counter value t.
- post(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.
- retrieve($t_{\text{start}}, t_{\text{end}}$, kw) $\to \{(v_i, t_i)\}$. It returns all pairs of (v_i, t_i), such that $t_{\text{start}} \le t_i \le t_{\text{end}}$ 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 $(t_{\text{start}}, t_{\text{end}}, \text{kw}')$ such that $t_{\text{start}} \le t \le t_{\text{end}}$ and $\text{kw}' = \text{kw}$ will receive a sequence of value/counter pairs containing (v, t).

Agreement. If an honest user retrieving with $(t_{\text{start}}, t_{\text{end}}, \text{kw})$ when getCounter() $\ge t_{\text{end}}$ 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. Compared with standard multicast, which is also for data dissemination, DDN saves communication costs when there are multiple nodes providing the same data block. Assuming there are m receivers out of n potential receivers, and there are k data providers for a data block of ℓ bits. Through multicast, every sender needs to send their data to every potential receiver, incurring 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 a total 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 [58] or practical infrastures like IPFS [67] to instantiate a DDN. In this work, we focus on the IPFS-based instantiation as it becomes 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 (bid, 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 $\mathcal{R}(s)$.

7.2 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.

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 data block 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. For 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.

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** $\mathsf{valid}_j = 0$

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

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

This ensures that any receivers who fail to receive the data through multicast will be able to retrieve 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' verification keys w.r.t. a digital signature scheme and VRF. A ratio $\mathsf{ratio}_{\mathsf{hm}}$ is also determined in the setup, which ensures a 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 on-chain storage cost is *independent of* ℓ.

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

Lemma 6. *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 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 that 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 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:

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

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 $|B \cap C| \geq t' + 1$, which implies the probability $\Pr[|B| \leq t]$ is small. Therefore, the adversary cannot corrupt all nodes in B even after knowing the committee C. It follows that there is always at least one honest node that has received v and provisioned it to the DDN, such that everyone can retrieve the data from the DDN and can agree on the value v. □

8 APPLICATION TO ALL-HANDS CHECKPOINTING INTO BITCOIN

In this section, we delineate how our DKG yields the first realization of the checkpointing blueprint Pikachu of Filecoin [2] that involves all validators in the whole blockchain network, e.g., Filecoin, that has 3700 of them, with various mining power.

8.1 Realizing the Bitcoin Checkpointing Pikachu with Any-Trust DKG

We review the checkpointing blueprint Pikachu in Appendix.A. At a high level, all validators need to run a DKG for Schnorr signature every epoch, and the resulting public keys will be used as Bitcoin addresses [10]. A Bitcoin transaction that embeds the digest of the PoS chain at epoch $i-1$ and transfers assets from the address at epoch $i-1$ to epoch i will serve as a checkpoint for epoch $i-1$. All validators jointly run the threshold Schnorr signing protocol to create such checkpointing transactions.

Pikachu only gave 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 [48], which relies on a coordinator, there may be a single point of failure. In the following, we demonstrate how our Any-Trust DKG can realize the blueprint efficiently and securely.

Sub-ID allocation. At each epoch i, the current validators of the blockchain locally run the deterministic sub-ID allocation algorithm on a publicly agreed power distribution, and then they obtain the same sub-ID allocation outcome. A validator with m sub-IDs will participate in further protocols as m individuals.

Our optimized sub-ID allocation algorithm in Sect.6 issues fewer sub-IDs to validators than the straightforward approach. We consider a snapshot of Filecoin's validator distribution[11], which has 3700 validators with a total mining power of around 25 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. The validators with 1688 sub-IDs will act like 1688 participants to execute the DKG protocol to generate a public key for Schnorr signature and share the secret keys. We set $\text{ratio}_{\text{at}} = 38/1688$, guaranteeing the committee has at least one good node with a probability of $1-5\times10^{-9}$. Then, the validators can run our Any-Trust DKG which incurs only around 3 MB of data that 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. At epoch i, the validators of epoch $i-1$ use their shared keys to sign the checkpointing Bitcoin transaction. Note that no matter how many nodes try to post the signed transaction to the Bitcoin, there will be only one transaction appearing on the chain. To sign this transaction, we adopt the GJKR protocol[35], which does not require a coordinator and is thus free of single-point 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 to 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 [43]), the static security of the resulting scheme directly follows the recent result in [61]. Note that despite recent advancements [24], achieving adaptively secure and robust threshold Schnorr without using a coordinator remains a significant open problem. We leave it as future work to analyze the adaptive security of this scheme, namely GJKR with an oracle-aided algebraic simulatable DKG.

[10] Bitcoin has supported Schnorr signature since its Taproot update.
[11] https://filfox.info/en/ranks/power

Table 4: Checkpointing cost per annum. in USD.

#Parties	2^7(Cosmos)	2^{10}(Polkadot)	2^{12}(Filecoin)
Babylon	1510826.9	2266245.6	6043306.5
Ours	26048.8		

*Based on the Bitcoin price on Mar. 31, 2024: 0.000708 USD per Satoshi.

8.2 Comparison with Babylon Checkpointing

Overview of Babylon. Babylon [63] 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 epoch number, the digest, the aggregated signature, and a bit vector that indicates the public keys involved.

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. Particularly, the epoch number, the block digest, and the aggregated signature together take 88 bytes; the bit-vector requires n bits. Therefore, the number of Bitcoin transactions for a Babylon checkpoint can be calculated as $\#\text{Bitcoin Tx}_{\text{Babylon}} = 1 + \lceil \frac{n+32}{640} \rceil$.

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 any-trust committee size κ, i.e., $\#\text{Bitcoin Tx}_{\text{secure-Babylon}} = \kappa + \kappa \cdot \lceil \frac{n+32}{640} \rceil$. For $\kappa = 29$ (see Table 2) and $n = 2^{12}$, we have $\#\text{Bitcoin Tx}_{\text{Babylon}} = 8$, while $\#\text{Bitcoin Tx}_{\text{secure-Babylon}} = 232$.

In comparison, our approach (Pikachu) only requires 1 Bitcoin transaction for each checkpoint, since the transaction is uniquely created via threshold signing, and Bitcoin will only accept one transaction no matter how many validators try to publish it. It is naturally free of single points of failure.

We compare the Bitcoin transaction fees for checkpointing per annum in Table 4, where the cost of Babylon is for its secure version. Following [63], we consider the checkpoint transactions to be created hourly. We assume, without loss of generality, each Bitcoin transaction has 300 bytes, the transaction fee is 14 Satoshi per byte (such that the transaction can be confirmed within six blocks as per [12]), and the price of a Satoshi is 0.000708 USD [13]. 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 [23]), 2^{10}validators for moderate-scale chains (like Polkadot[57]), and 2^{12} validators for relatively large-scale chains (like Filecoin [31]).

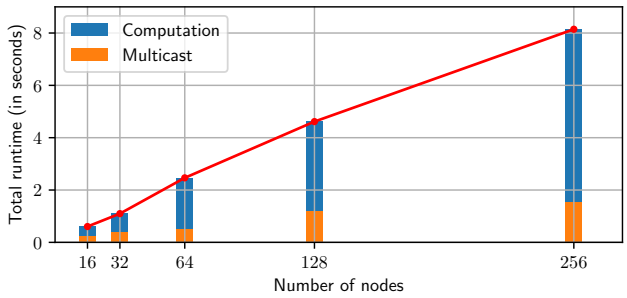
9 IMPLEMENTATION AND EVALUATION

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

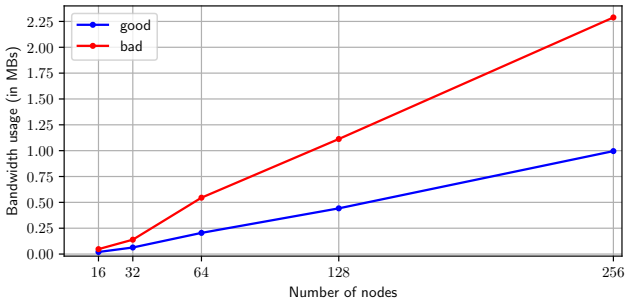
[12]https://btc.network/estimate

[13]updated on Mar. 31, 2024, from https://coincodex.com/crypto/satoshi-sats/

Implementation. We implemented our protocol in Java 8, comprising approximately 1500 lines of code. To facilitate Elliptic Curve operations and communication, we utilized the open-source Java library `mpc4j`[14] and the `Bouncy Castle` library[15]. Given our protocol's primary application in creating checkpoints on Bitcoin, we opted for the secp256k1 curve and SHA-256 for cryptographic operations. Our implementation includes components such as VRF and multi-recipient encryption but does not employ the broadcast extension trick in Appendix 7. 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. Whenever possible, we set the expected size s of an any-trust group to 38, which ensures that the committee qualifies with a probability of $1 - 5 \times 10^{-9}$, as in Table 2. For small-scale tests like $n = 16$ and 32, we set $s = n/2 + 1$.



(a) Worst-case Adjusted Running Time of bad instances.



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

Figure 3: End-to-end Test Results

9.1 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 machine (VM). Each VM has 2 vCPUs and 4 GiB RAM and runs in Amazon Linux 2023 AMI 2023.4.20240416.0 x86_64 HVM kernel-6.1. All nodes are placed in the same AWS region and are connected pair-wise; for example, every two nodes are directly connected. Since the network delay within the same AWS region is almost negligible, we simulate a

[14]https://github.com/alibaba-edu/mpc4j

[15]https://www.bouncycastle.org/

more realistic delay by employing the Linux command `tc` (traffic control) to introduce an artificial delay of 100 ms for all traffic.

Implementation Remarks. We set up an additional node to simulate a blockchain, which serves as the broadcast channel in our implementation. The blockchain node is directly connected to all other nodes. In Round 1 and Round 3 of our DKG, whenever a node needs to broadcast a message, it sends the message directly to the blockchain node. The blockchain node then relays all received messages in the round to every node in the network. As our protocol assumes network synchrony and proceeds round by round, we need to specify the time window for each round.

In practice, the time window setting for Round 1 and 3 can vary depending on the blockchain. For simplicity, we artificially configure the time window to be 30 seconds: the blockchain node receives messages in the first 20 seconds and then relays the messages. All nodes stop receiving current-round messages at 30 seconds and move to the next round. Given such a configuration, a 60-second broadcast running time is inherent to our experiments, and our experiments are more concerned about the running time incurred by Round 2 and the computation cost in Round 1, Round 3 and at the end of Round 3.

We evaluate the performance of our DKG in both the good-case and bad-case scenarios. In the good cases, all nodes are honest. In the bad cases, we set all nodes whose node ID is smaller than $n/2$ to be corrupted. A corrupted node, if elected as a dealer in Round 1, will broadcast malformed ciphertexts to all nodes, causing n complaints against it in Round 2. Given a fixed number of nodes and good or bad cases, each experiment configuration is repeated eight times.

In reality, the timeout parameter for the **receive{}** at the start of Round 3 should be calibrated based on the communication cost in bad cases. In our implementation, the calibration is implicit: the timeout parameter is set to a sufficiently large value, while the actual communication cost in bad cases is measured independently.

Adjusted Running Time. The total running time of the entire Any-Trust DKG protocol can be defined by 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. However, this measurement will always incorporate the 60-second broadcast time, which may vary in different settings. Hence, an adjusted running time is measured by subtracting the 60-second broadcast cost from the running time of the whole Any-Trust DKG protocol. Observe that the adjusted running time consists only two components: the communication cost incurred by the **multicast** in Round 2, and all computation costs throughout this protocol.

We take the **maximum** adjusted running time across all nodes, and all repeats, to represent the end-to-end running time of our protocol. The adjusted running time of bad cases are shown in Fig.3a. In additional, a breakdown by the **multicast** communication cost and the computation cost is shown within the stacked bar chart. Note that the good cases should always perform better than the bad cases, hence, we only represent the bad cases to demonstrate the worst-cast scenario.

Our DKG protocol only requires a few seconds to finish the multicast round and all computation tasks, in additional to the omitted 60-second broadcast cost.

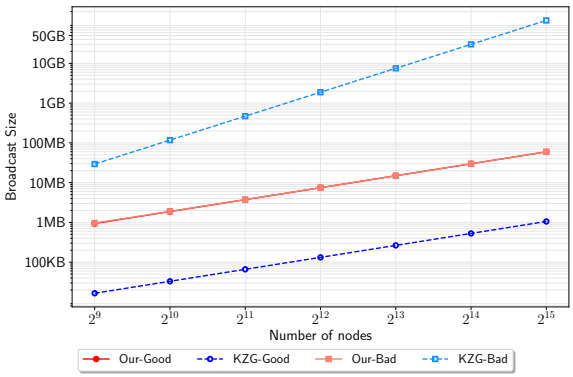


Figure 4: Broadcast Channel Overhead

Bandwidth Usage. We record the inbound and outbound bandwidth of each node in Megabytes (10^6 bytes) and demonstrate the **maximum** bandwidth usage of all nodes, and all repeats in Fig.3b. The key observation is that the bandwidth grows linearly depending on the size of the group.

At first glance at the results, some non-linearity may be noticed. However, this is mainly caused by (a) a lower sortition ratio for $n = 16$ and $n = 32$ and (b) the randomness in the sortition results in the Any-Trust DKG protocol. Specifically, the protocol has no deterministic control over the actual number of parties being elected as dealers, meaning fluctuations will be observed in bandwidth usage, as the actual number of dealer may vary.

9.2 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.

Broadacst cost. We calculate the total number of bits to be sent via the broadcast channel. We compare our protocol and 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. Note that in KZG, a share along with the proof for validating has the size of 224 Bytes; in the bad case, there are $n^2/2$ shares (with their proofs) to be broadcasted for public verification.

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 960 KB, while for $n = 2^{15}$, the cost is approximately 59.6 MB. In contrast, while the good-case KZG has very low broadcast costs, its worst-case costs grow quadratically and would require over 120 GB when $n = 2^{15}$.

Computation time. We conducted tests to measure the computation time for generating a secret-sharing transcript (Deal) and reaching an 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 [70] for the good case while estimating the worst-case scenario by assuming that $n^2/2$ shares need to be verified (each share verification takes 1.3 ms). As illustrated in Fig.5, in the good case, our protocol's performance is comparable to or even better than KZG, although their programming language (C++)

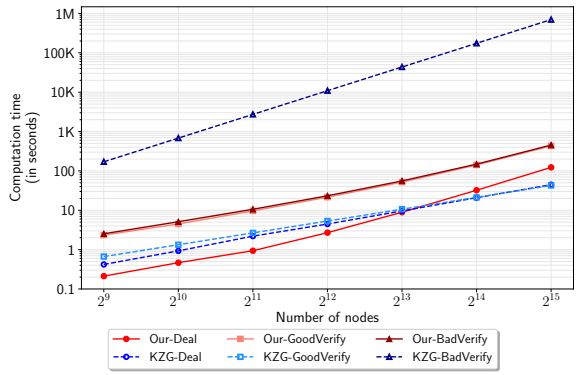


Figure 5: Computation Overhead

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 naïve 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 [70] 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.

10 RELATED WORKS

VSS-based DKG. Distributed Key Generation (DKG) has been a prominent area of research for several decades. Pedersen's seminal work [56] 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) [29]. 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. [46], 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 $O(n^2)$ [65]. However, a recent exploration by Zhang et al. [70] 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 [69, 70] that prioritize efficient polynomial commitments without relying on a trusted setup, but these don't match KZG's efficiency.

PVSS-based DKG. Fouque and Stern [32] 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 $O(n\mathrm{BB}_n(n\lambda))$ 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 [17], 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 applied to improve many VSS-based DKG schemes, 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 few recent works focus on improving the concrete performance of PVSS schemes, including the lattice-based PVSS [37], Groth's PVSS [41], and PVSS using class groups [45].

Aggregatable-PVSS-based DKG. Aggregatable PVSS schemes [43] are PVSS schemes whose transcripts can be concisely merged into one. There are a few designs that leverage customized communication protocols rather than simply leveraging Byzantine broadcast protocols (or broadcast channels), enjoying asymptotically better complexity. Notably, Gurkan et al. [43] 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. Very recently, Feng et al.[30] and Bacho et al. [3] leverage specially designed communication protocols together with aggregatable PVSS schemes and present DKG schemes with sub-quadratic per-party computation/communication cost while enjoying optimal resilience.

Note that existing aggregatable PVSS schemes all produce secrets in an Elliptic curve group, thus incompatible with many threshold cryptographic protocols. Feng et al. [30] also give a variant of DKG

using conventional PVSS schemes but with slightly higher (still sub-quadratic) per-party complexity.

DKG in the YOSO model. A common strategy to enhance scalability is selecting a committee and executing the threshold cryptographic systems within this smaller subset. However, this approach is fraught with challenges. Once aware of the committee's composition, an adaptive adversary 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 [21]) and assuming memory erasure fail to provide protection against adaptive attackers. Recent advances in the YOSO (You-Only-Speak-Once) MPC realm [10, 36] hint at potential solutions to deter adaptive adversaries targeting the committee. Benhamouda et al. [10] presents a DKG in the YOSO model. However, the YOSO techniques come with their own set of challenges. Notably, existing YOSO techniques (if without using resource-intensive tools like fully homomorphic encryption [38]) need to sample a huge committee, say with a few or tens of thousands of nodes, which is already as large as the network scale we are interested in, let alone the extra overhead incurred by using YOSO techniques. Furthermore, as successive committees remain anonymous, inter-committee communication is heavily dependent on a broadcast channel.

On the security of DKG. Beyond endeavors aimed at bolstering the efficiency of DKG, various research initiatives have tackled this challenge from different perspectives. Gennaro et al. [35] 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. [43] 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 [56], Fouque-Stern's [32], and our own, fitting this criteria. Another remarkable contribution by Canetti et al. [16] 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 [4] put forth an oracle-aided adaptive definition and ascertained that several protocols, including [32, 56], conform to this definition in the algebraic group model. Our model also complies with this adaptive security definition.

Asynchronous DKG. Lastly, some recent research efforts [1, 25, 33] 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. [25] showcased the inaugural asynchronous DKG with a communication overhead of $O(n^3\lambda)$ for field-element secrets, whereas Abraham et al. [1] furnished an adaptively secure asynchronous DKG with identical complexity.

REFERENCES

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

[2] Sarah Azouvi and Marko Vukolic. 2022. Pikachu: Securing PoS Blockchains from Long-Range Attacks by Checkpointing into Bitcoin PoW using Taproot. In *ConsensusDay@CCS*. ACM, 53–65.

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

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

[5] Renas Bacho and Julian Loss. 2023. Adaptively Secure (Aggregatable) PVSS and Application to Distributed Randomness Beacons. In *CCS*. ACM, 1791–1804.

[6] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. 2018. Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability. In *CCS*. ACM, 913–930.

[7] Mihir Bellare, Alexandra Boldyreva, Kaoru Kurosawa, and Jessica Staddon. 2007. Multirecipient Encryption Schemes: How to Save on Bandwidth and Computation Without Sacrificing Security. *IEEE Trans. Inf. Theory* 53, 11 (2007), 3927–3943.

[8] Mihir Bellare, Elizabeth C. Crites, Chelsea Komlo, Mary Maller, Stefano Tessaro, and Chenzhi Zhu. 2022. Better than Advertised Security for Non-interactive Threshold Signatures. In *CRYPTO (4) (LNCS, Vol. 13510)*. Springer, 517–550.

[9] Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. 2020. Can a Public Blockchain Keep a Secret?. In *TCC (1) (LNCS, Vol. 12550)*. Springer, 260–290.

[10] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. 2022. Threshold Cryptography as a Service (in the Multiserver and YOSO Models). In *CCS*. ACM, 323–336.

[11] Aptos Blockchain. [n. d.]. Aptosan. https://aptoscan.com.

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *ASIACRYPT (LNCS, Vol. 2248)*. Springer, 514–532.

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

[14] Carlo Brunetta, Hans Heum, and Martijn Stam. 2024. SoK: Public Key Encryption with Openings. In *PKC (4) (LNCS, Vol. 14604)*. Springer, 35–68.

[15] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *J. Cryptol.* 18, 3 (2005), 219–246.

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

[17] Ignacio Cascudo and Bernardo David. 2017. SCRAPE: Scalable Randomness Attested by Public Entities. In *ACNS (LNCS, Vol. 10355)*. Springer, 537–556.

[18] Andrea Cerulli, Aisling Connolly, Gregory Neven, Franz-Stefan Preiss, and Victor Shoup. 2023. vetKeys: How a Blockchain Can Keep Many Secrets. Cryptology ePrint Archive, Paper 2023/616. https://eprint.iacr.org/2023/616.

[19] Melissa Chase and Anna Lysyanskaya. 2006. On Signatures of Knowledge. In *CRYPTO (LNCS, Vol. 4117)*. Springer, 78–96.

[20] David Chaum and Torben P. Pedersen. 1992. Wallet Databases with Observers. In *CRYPTO (LNCS, Vol. 740)*. Springer, 89–105.

[21] Jing Chen and Silvio Micali. 2019. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.* 777 (2019), 155–183.

[22] Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. SoK: Distributed Randomness Beacons. In *SP*. IEEE, 75–92.

[23] Cosmos. [n. d.]. https://cosmos.network.

[24] Elizabeth C. Crites, Chelsea Komlo, and Mary Maller. 2023. Fully Adaptive Schnorr Threshold Signatures. In *CRYPTO (1) (LNCS, Vol. 14081)*. Springer, 678–709.

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

[26] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. In *EUROCRYPT (2) (LNCS, Vol. 10821)*. Springer, 66–98.

[27] Luciano Freitas de Souza and Andrei Tonkikh. 2023. Swiper and Dora: efficient solutions to weighted distributed problems. *CoRR* abs/2307.15561 (2023).

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

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

[30] Hanwen Feng, Zhenliang Lu, and Qiang Tang. 2023. Breaking the Cubic Barrier: Distributed Key and Randomness Generation through Deterministic Sharding. Cryptology ePrint Archive, Paper 2024/168. https://eprint.iacr.org/2024/168.

[31] Filecoin. [n. d.]. https://filecoin.io/.

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

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

[34] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT (2) (LNCS, Vol. 9057)*. Springer, 281–310.
[35] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. *J. Cryptol.* 20, 1 (2007), 51–83.
[36] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. 2021. YOSO: You Only Speak Once - Secure MPC with Stateless Ephemeral Roles. In *CRYPTO (2) (LNCS, Vol. 12826)*. Springer, 64–93.
[37] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. 2022. Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties. In *EUROCRYPT (1) (LNCS, Vol. 13275)*. Springer, 458–487.
[38] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. 2021. Random-Index PIR and Applications. In *TCC (3) (LNCS, Vol. 13044)*. Springer, 32–61.
[39] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP*. ACM, 51–68.
[40] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, and Leonid Reyzin. 2016. NSEC5 from Elliptic Curves: Provably Preventing DNSSEC Zone Enumeration with Shorter Responses. *IACR Cryptol. ePrint Arch.* (2016), 83.
[41] Jens Groth. 2023. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Paper 2021/339. https://eprint.iacr.org/2021/339.
[42] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. In *CCS*. ACM, 803–818.
[43] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Aggregatable Distributed Key Generation. In *EUROCRYPT (1) (LNCS, Vol. 12696)*. Springer, 147–176.
[44] Gene Itkis and Leonid Reyzin. 2001. Forward-Secure Signatures with Optimal Signing and Verifying. In *CRYPTO (LNCS, Vol. 2139)*. Springer, 332–354.
[45] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. 2023. Non-interactive VSS using Class Groups and Application to DKG. Cryptology ePrint Archive, Paper 2023/451. https://eprint.iacr.org/2023/451.
[46] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT (LNCS, Vol. 6477)*. Springer, 177–194.
[47] Dafna Kidron and Yehuda Lindell. 2011. Impossibility Results for Universal Composability in Public-Key Models and with Fixed Inputs. *J. Cryptol.* 24, 3 (2011), 517–544.
[48] Chelsea Komlo and Ian Goldberg. 2020. FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In *SAC (LNCS, Vol. 12804)*. Springer, 34–65.
[49] Sung-Shine Lee, Alexandr Murashkin, Martin Derka, and Jan Gorzny. 2023. SoK: Not Quite Water Under the Bridge: Review of Cross-Chain Bridge Hacks. In *ICBC*. IEEE, 1–14.
[50] Dahlia Malkhi and Pawel Szalachowski. 2022. Maximal Extractable Value (MEV) Protection on a DAG. In *Tokenomics (OASIcs, Vol. 110)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 6:1–6:17.
[51] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. 1999. Verifiable Random Functions. In *FOCS*. IEEE Computer Society, 120–130.
[52] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The Honey Badger of BFT Protocols. In *CCS*. ACM, 31–42.
[53] Moni Naor and Moti Yung. 1990. Public-key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. In *STOC*. ACM, 427–437.
[54] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *DISC (LIPIcs, Vol. 179)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:17.
[55] Jesper Buus Nielsen. 2002. Separating Random Oracle Proofs from Complexity Theoretic Proofs: The Non-committing Encryption Case. In *CRYPTO (LNCS, Vol. 2442)*. Springer, 111–126.
[56] Torben P. Pedersen. 1991. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *CRYPTO (LNCS, Vol. 576)*. Springer, 129–140.
[57] Polkadot. [n. d.]. https://www.polkadot.network/.
[58] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.
[59] Ruby.Exchange. 2021. How SKALE Solves The Front-Running Problem. https://blog.ruby.exchange/how-skale-solves-the-front-running-problem/?ref=blog.pantherprotocol.io.
[60] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. 2001. Robust Non-interactive Zero Knowledge. In *CRYPTO (LNCS, Vol. 2139)*. Springer, 566–598.
[61] Victor Shoup. 2023. The many faces of Schnorr. Cryptology ePrint Archive, Paper 2023/1019. https://eprint.iacr.org/2023/1019.
[62] Selma Steinhoff, Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolic. 2021. BMS: Secure Decentralized Reconfiguration for Blockchain and BFT Systems. *CoRR* abs/2109.03913 (2021).
[63] Ertem Nusret Tas, David Tse, Fangyu Gai, Sreeram Kannan, Mohammad Ali Maddah-Ali, and Fisher Yu. 2023. Bitcoin-Enhanced Proof-of-Stake Security: Possibilities and Impossibilities. In *SP*. IEEE, 126–145.
[64] Tezos. [n. d.]. https://tezos.com.
[65] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. 2020. Towards Scalable Threshold Cryptosystems. In *IEEE Symposium on Security and Privacy*. IEEE, 877–893.
[66] Total-blockchain. 2022. Osmosis will soon be frontrunning MEV free. https://medium.com/@totalblockchainemail/osmosis-will-soon-be-frontrunning-mev-free-b7da89f04ce9.
[67] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. 2022. Design and evaluation of IPFS: a storage layer for the decentralized web. In *SIGCOMM*. ACM, 739–752.
[68] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. 2012. Scalable anonymous group communication in the anytrust model. In *European Workshop on System Security (EuroSec)*, Vol. 4.
[69] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. 2022. hbACSS: How to Robustly Share Many Secrets. In *NDSS*. The Internet Society.
[70] Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. 2022. Polynomial Commitment with a One-to-Many Prover and Applications. In *USENIX Security Symposium*. USENIX Association, 2965–2982.

A SUPPLEMENTARY MATERIALS FOR BITCOIN CHECKPOINTING

A.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 malicious forking by punishing misbehavior 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 late coming client may not be able to decide the canonical chain among the forks.

Securing PoS with Bitcoin checkpointing. A few works [2, 63] 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 coming 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 transfers all assets on Q_i to Q_{i+1}, the address belongs to the current configuration C_{i+1}; This transaction is the checkpoint. We elucidate their design with the following three algorithms/protocols[16].

- AllocateSubID$(C) \rightarrow \{d_j\}_{j\in[n]}$.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.

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

- 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 $\vec{pk} = (pk_{j,z})_{j\in[n],z\in[d_j]}$.
- CreateCKP(C_i, ckp, PreAdd, Q_{i+1}) $\rightarrow$ 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 invoke a Threshold Schnorr protocol to sign a Bitcoin transaction TX with the following 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 block tree of finalized PoS blocks. Specifically, the user first synchronizes with the 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 block whose digest is ckp as the canonical chain. Moreover, while other approaches like key-evolving forward-secure signatures [21, 26] may also mitigate long-range attacks, the checkpointing mechanism enjoys the unique advantage of ensuring malicious validators are always slashable. We defer a detailed discussion to Sect.A.2.

A.2 Security of Checkpointing

This paradigm has been thoroughly analyzed in [2]. 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 that 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 7 ([2]). *Assume both the Bitcoin blockchain and the PoS chain satisfy consistency, chain growth, and chain quality (as defined in [34]). 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-IDs 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 safety. 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' stakes for a period and slashing one's stake once 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 [63] that slashable safety against long-range attacks is impossible without external trust. With this result, [63] also shows that other approaches for mitigating long-range attacks, such as key-evolving signatures [6, 21] cannot provide slashable safety. Nonetheless, leveraging the Bitcoin blockchain as an external trust can certainly bypass this impossibility. Assuming that checkpoints for the canonical PoS chain have been properly posted on the Bitcoin blockchain, the attacker cannot present an attack chain that diverges from the canonical chain before the latest checkpoint. In this case, the attacker must not have withdrawn its stakes and thus is slashable.

In light of the above, both ours/Pikachu and Babylon can guarantee slashable safety once the checkpoints have been properly created. Now, we turn to examine the case in which checkpoints may not be generated correctly. The adversary has the following options: (1) not make a checkpoint; (2) make a checkpoint for an ill-formed block; (3) make more than one checkpoint 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 exclude 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 process this chain. In case the adversary refuses to participate in the checkpoint creation, Babylon considered the punishment of inactivity, which enables the removal of the inactive validators. Regarding censorship resistance (4), Babylon proposed a roll-up technique that 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 blockchain, 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 breaks.