Scalable Distributed Key Generation for Blockchains

Abstract—**Distributed key generation (DKG) is a foundational building block for designing efficient threshold cryptosystems, which are crucial components of blockchain ecosystems. Existing DKG protocols address the problem in a standalone setting, focusing on establishing the final DKG public key and individual secret keys among the participating parties. This work focuses on DKG primitives for use over blockchain, where the final DKG public key must be available on-chain, enabling on-chain smart contracts to seamlessly execute threshold cryptographic verifications. We observe that existing standalone DKG designs do *not* sufficiently exploit the presence of blockchain, leaving substantial scope for improvement in performance.**

In this work, we design the first discrete-log-based DKG protocol tailored for use over blockchain, leveraging the blockchain's built-in consensus mechanism to realize DKG efficiently. Interestingly, the use of blockchains enables us to solve DKG while tolerating up to one-half Byzantine faults even in non-synchronous settings. Our protocol is asynchronous, allowing it to operate independently of the network's timing assumptions, with the exact network model depending on the destination blockchain.

Our solution further utilizes an associated random beacon to select smaller committees and achieves a DKG protocol with sub-cubic communication complexity, sub-quadratic computation complexity, and minimal on-chain storage. Notably, our protocol employs a single invocation of consensus and can terminate in just eleven communication rounds in the good case when deployed on an optimal latency partially synchronous blockchain. Our experiments show that our protocol terminates faster than state-of-the-art standalone protocols, with similar bandwidth overhead for committee members and significantly reduced bandwidth for other parties. Additionally, our protocol benefits from higher CPU resources—when deployed on machines with 32 vCPUs, it completes in approximately 6.5 seconds in the optimistic case, even for larger systems with 256 nodes.

1. Introduction

Distributed key generation (DKG) [47] aims to establish a public key and its corresponding secret keys among a group of n parties in a secret-sharing fashion such that no single party knows the shared secret key. DKG protocols reduce trust assumptions in cryptographic systems such as threshold signatures [15] and threshold encryption schemes [34]. These threshold cryptosystems have diverse applications, including implementing random beacons [19], [37], simplifying consensus protocols [75], [80], enabling secure multiparty computation [54], [55], and delegating secret management to multiple semi-trusted authorities [39], [61].

As threshold cryptography gains traction in blockchains and distributed systems, DKG has emerged as a topic of significant interest. A substantial body of research has focused on addressing the DKG problem efficiently, with efforts dedicated to reducing communication and round complexity while ensuring optimal resilience across various network settings [1], [3], [11], [29], [32], [41], [47], [56], [76]. However, to the best of our knowledge, these protocols focus on solving the DKG problem in a standalone context, where the primary goal is to agree on the final DKG public key and corresponding individual secret keys among n parties.

Distributed key generation for use over a blockchain. With the growing adoption of blockchains, several on-(block)chain applications (e.g., generating or/and verifying randomness [37], [49], [53] and state commitments) demand the ability to validate threshold signatures directly on-chain. In particular, on-chain smart contracts often require access to the final DKG public key to facilitate the verification of threshold signatures within the blockchain. To support these applications, it is essential to extend the DKG primitive by defining *Distributed key generation for use over blockchains*, which mandates that the final DKG public key generated through the DKG process be recorded on-chain—this is necessary for on-chain verification of threshold signatures, beacons, etc. Moreover, the DKG protocol should not impose any additional network assumptions on the (destination) blockchain that uses it. In particular, the design must be asynchronous to ensure that the network assumptions of the employed blockchain system remain unchanged.

A straightforward approach to address this problem is to first execute an existing standalone asynchronous DKG (ADKG) protocol and subsequently publish the signed DKG public key on-chain. The blockchain then performs a total ordering of the posted signed DKG public keys, allowing a common final DKG public key, signed by at least $f + 1$ parties, to be used on-chain, where f is the maximum number of Byzantine parties.

While this approach is conceptually simple, it incurs significant overhead in terms of communication, computation, and round complexity, and it restricts the protocol's fault tolerance to $f < n/3$. Specifically, the participating parties must first execute a standalone ADKG protocol, which requires a separate (asynchronous) consensus instance with $n > 3f$ to agree on a common set of parties that have correctly shared their secrets. Following this, when the signed DKG public key is posted on-chain, the process involves yet

TABLE 1: **Comparison of High-Threshold DKGs for use over Blockchains**

	Res.	**Off-chain Comm.**	**On-chain Storage**	**Off-chain Computation**	**Round**	**Setup**
Kokoris et al. [60]	1/3	$O(\kappa n^4)$	$O(\kappa n)$	$O(n^3)$	$E(O(n))$	PKI
Das et al. [32]	1/3	$O(\kappa n^3)$	$O(\kappa n)$	$O(n^2)$	$E(O(\log n))$	PKI
Das et al. [29]	1/3	$O(\kappa n^3)$	$O(\kappa n)$	$O(n^2)$	$E(O(\log n))$	PKI
Abraham et al. [3]	1/3	$O(\kappa n^3)$	$O(\kappa n)$	$O(n^2)$	$E(O(1))$	PKI
Zhang et al. [82]	1/3	$O(\kappa n^4)$	$O(\kappa n)$	$O(n^3)$	$E(O(2^n))$	-
Kate et al. [58] Cascudo et al. [22]	1/2	-	$O(\kappa n^2)$	$O(n^2)$	5	PKI
Feng et al. [43]	1/3	$O(\kappa n^2)$	$O(\kappa n)$	$O(n^2)$	$E(O(1))$	PKI+SS
This work	1/2	$O(\kappa n^3 + n_a n^3)$	$O(\kappa n_a + n_a n)$	$O(n^2)$	10	PKI+RB
This work	1/3	$O(\kappa n_c^2 n + n_a n_c^3 + \kappa n^2)$	$O(\kappa n_a + n_a n_c)$	$O(n_c)/O(n_c n)^*$	11	PKI+RB

In this table, we only compare DKG protocols that generate scalar secret keys. Our protocol leverages random beacons to select smaller committees and improve efficiency. We argue that, even with access to random beacons and blockchain, existing DKG protocols cannot achieve the same level of efficiency as our protocol, as detailed in Section 3. **Res.** implies resilience. **Comm.** implies communication. **E(.)** implies "in expectation". **SS** denotes a silent setup phase, which incurs $O(\kappa n^3)$ communication overhead. **RB** implies random beacon. * $O(n_c)/O(n_c n)$ implies that parties outside the clan perform $O(n_c)$ computations, while parties within the clan perform $O(n_c n)$ computations.

another consensus instance, as the blockchain nodes order the blocks. Overall, this straightforward approach inherently involves two separate consensus instances, which, as we discover, is unnecessary. This redundancy not only adds substantial communication overhead and increases round complexity but also limits the resilience of the DKG process to tolerate at most $f < n/3$ Byzantine faults, as opposed to up to the best possible $f < n/2$ Byzantine faults (as we demonstrate) for DKG/threshold cryptography.

One may also think about using the recent non-interactive publicly verifiable secret sharing (PVSS) schemes [22], [46], [50], [58] to realize DKG for use over a blockchain. In this approach, every party posts their PVSS vector onto the blockchain, and the first set of $f + 1$ valid PVSS vectors can be used to establish the DKG[1]. This method is efficient in terms of off-chain communication and can tolerate up to $n/2$ Byzantine parties within the DKG committee. However, each PVSS vector has a size of $\Omega(\kappa n)$, which leads to the blockchain needing to store at least $O(\kappa n^2)$ information, plus also incurs $O(n^2)$ computation, where κ is the security parameter and n is the number of parties in the DKG committee. This approach becomes non-scalable due to the high gas fees associated with storing and verifying $O(n^2)$ data on the blockchain.

Finally, publicly verifiable and unpredictable random

1. We note that non-interactive PVSS approaches [22], [46], [50], [58] have been utilized to realize DKG assuming broadcast channels. We observe that, due to the non-interactive nature of these schemes, they can also be employed to realize DKG in a non-synchronous blockchain setting, despite the blockchain offering weaker atomic broadcast [24] guarantees compared to broadcast channels: a broadcast channel should ensure that every honest input appears on the channel in within a know time bound, which we cannot ensure on any populer blockchain without setting time-bound to be extremely large. For the lightning network over Bitcoin, they set this time-bound to be in days.

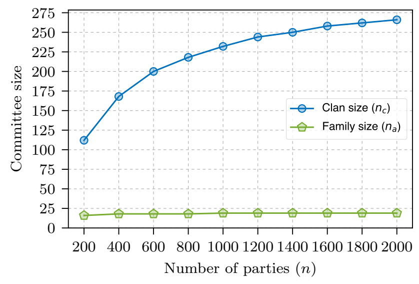

Figure 1: Committee sizes needed for a clan and a family as a function of n ensuring a failure probability $< 10^{-9}$.

beacons [37], [53], [63], [70] are readily available as a service and can be used to construct efficient DKG protocols (as in prior work [42]). For example, these random beacons can be utilized to select smaller committees of dealers who perform the DKG, enhancing the efficiency significantly. However, we observe that even with access to random beacons and blockchain, existing standalone DKG protocols remain inefficient (as explained later in Section 3 when we provide more context). This naturally begets the following important question:

> Leveraging access to blockchain and random beacons can we design a significantly more efficient discrete-log-based DKG protocol for blockchains?

We affirmatively address this question by introducing the first discrete-log based DKG protocol tailored for blockchain equipped with a randomness beacon. By using the beacon, we randomly sample sub-committees and design efficient DKG protocols under two fault tolerance settings: super-honest majority ($f < n/3$) and honest majority ($f < n/2$). In the super-honest majority setting, our protocol achieves $O(\kappa n_c^2 n + n_a n_c^3 + \kappa n^2)$ communication and $O(\kappa n_a + n_a n_c)$ on-chain storage. Here n_c denotes the size of an honest-majority sub-committee (termed *clan*) sampled from n parties, while n_a refers to a smaller sub-committee (termed *family*) with at least one honest party, also sampled from n parties. Comparative committee sizes are shown in Figure 1. In this setting, the computational load is primarily handled by the clan, performing $O(n_c n)$ computation, while the remaining parties carry out only $O(n_c)$ computations each. Importantly, our protocol requires only a single invocation of the consensus mechanism–realized by blockchain and completes in 11 rounds in the good-case[2] when using an optimal latency partially synchronous blockchain [35], [74]. This setting is tailored for applications like consensus and on-chain randomness, where a super-honest majority assumption is necessary. In the honest majority setting, our protocol achieves $O(\kappa n^3 + n_a n^3)$ communication, $O(\kappa n_a + n_a n)$ on-chain storage, $O(n^2)$ off-chain computation and terminate in 10 rounds in the good-case when using an optimal-latency partially synchronous blockchain protocol [35], [74].

Finally, our protocol generates scalar secret keys and is compatible with off-the-shelf threshold cryptosystems [16],

2. Good-case implies the leader of the underlying consensus protocol is honest, and the network is synchronous. [6]

[34]. We present a comparison of our protocol with existing approaches for DKG over blockchain in Table 1.

Implementation and evaluation. We implemented and evaluated our protocol, comparing its performance to the state-of-the-art standalone DKG by Das et al. [29] when deployed over a blockchain. At $n = 256$, our protocol terminates approximately $3.3\times$ faster in optimistic scenarios, where all parties respond promptly, and about 17% faster during pessimistic executions with up to f Byzantine failures when deployed on machines with 2 vCPUs. With higher computational resources, the efficiency improves significantly—on machines with 32 vCPUs, our protocol completes in approximately 6.5 seconds in optimistic conditions, demonstrating its scalability. Additionally, the bandwidth consumption for clan parties is approximately $3.8\times$ lower in optimistic cases and comparable in pessimistic cases. We also computed the Ethereum gas cost for posting data on-chain when using Ethereum as a blockchain and found that our protocol incurs significantly lower gas costs compared to those of Kate et al. [58] and Das et al. [29] due to minimal on-chain storage required by our protocol.

Organization. We review related works in Section 2, followed by a technical overview in Section 3. The system model and preliminaries are presented in Section 4. Our proposed protocol is detailed in Section 6, with its evaluation in Section 7.

2. Related Work

An extensive body of research has focused on enhancing the performance of DKG protocols under various network settings. To the best of our knowledge, existing DKG protocols primarily operate in a standalone context and do not account for ensuring that the final DKG public key appears on the blockchain. In this work, we present the first dlog-based DKG protocol designed specifically for use over blockchain, leveraging the blockchain's built-in consensus mechanism and random beacons to achieve efficient DKG. We review the most recent and closely related works below. We detail synchronous DKG protocols in Appendix B.

Non-interactive VSS and DKG. We extensively compared non-interactive VSS and DKG approaches in the introduction and highlighted their inefficiencies even when provided with access to blockchain and random beacons.

Any-trust DKG [42]. Feng et al. [42] proposed a synchronous DKG protocol in the "any-trust" model, where they select a much smaller committee (similar to our family) with the guarantee that at least one party in the committee is honest, except with negligible probability of error. In the synchronous setting, honest messages are guaranteed to arrive in a timely manner, enabling the selection of a significantly smaller committee. Like our work, they assume random beacons for electing a committee of dealers.

They further assume a synchronous blockchain and use it as a broadcast channel. To optimize on-chain storage, they leverage external storage systems such as IPFS [79], posting only metadata on the blockchain. However, since practical blockchains do not provide synchrony guarantees, their approach becomes impractical in real-world settings. Additionally, their work does not address the final public key appearing on the blockchain.

Partially synchronous and asynchronous DKG protocols. Kate et al. [57] proposed the first practical DKG protocol in the partially synchronous communication model, achieving $O(\kappa n^4)$ communication complexity and $O(n)$ round complexity. Kokoris-Kogias et al. [60] introduced the first ADKG protocol with optimal resilience, also featuring $O(\kappa n^4)$ communication complexity and $O(n)$ round overhead.

Abraham et al. [4] improved upon this by presenting a protocol with $O(\kappa n^3)$ communication and expected $O(1)$ round complexity. However, their protocol does not support key generation for discrete-log-based cryptosystems and relies on stronger cryptographic assumptions, such as SXDH. Das et al. [32] extended the research by designing a discrete-log-based DKG protocol with $O(\kappa n^3)$ communication and optimal resilience in the asynchronous model, albeit with expected $O(\log n)$ round complexity and requiring the Decisional Composite Residuosity (DCR) assumption. In a follow-up work, Das et al. [29] presented another discrete-log-based DKG protocol with $O(\kappa n^3)$ communication, optimal resilience in the asynchronous model, and a weaker CDH assumption. However, this protocol still incurred an expected $O(\log n)$ round complexity.

More recently, Abraham et al. [3] developed an adaptively secure discrete-log-based DKG protocol with $O(\kappa n^3)$ communication and expected $O(1)$ constant rounds. However, the security of this protocol depends on stronger cryptographic assumptions, including the one-more discrete logarithm (OMDL) assumption and the non-standard algebraic group model (AGM).

Zhang et al. [82] proposed a DKG protocol with minimal assumptions, eliminating the need for PKI and relying solely on the discrete logarithm (DL) assumption. However, their protocol incurs $O(\kappa n^4)$ communication complexity and exhibits exponential runtime in the worst case due to its dependence on local coins for randomness.

As mentioned earlier, all these works focus on the standalone setting, where the final public key is not posted on the blockchain. Furthermore, as discussed in Section 3, even with access to random beacons and blockchain, these approaches cannot achieve the same efficiency as our protocol.

We also note a closely related work on asynchronous proactive secret sharing (APSS) by Gunther et al. [51]. Their solution relies on computationally heavy ACSS primitives like Haven [8] and single-shot VABA [5] for consensus and incurs $O(\kappa n^3)$ communication and expected $O(1)$ rounds. They also propose a committee-based variant with sub-cubic communication by electing an honest-majority sub-committee, but it requires $O(\log n)$ rounds—unlike Das et al. [29], which we evaluate against and which achieves coin-free, constant-round execution in the good-case. Their protocol thus underperforms Das et al. [29] and additionally overlooks on-chain storage, a key consideration in blockchain applications.

3. Technical Overview

Standalone DKG and strawman alternatives. Existing standalone ADKG protocols [3], [29], [32], [82] commonly employ n parallel asynchronous verifiable/complete secret sharings (AVSS/ACSS) [40], [68], [81]. An ACSS scheme allows a dealer to distribute secret shares to parties such that (i) each party can verify the consistency of their received share with the shared secret and (ii) a subset of parties can reconstruct the secret by combining their shares. These protocols typically utilize weaker consensus primitives, such as reliable broadcast [18] (RBC), to ensure that the dealer distributes a consistent secret to all honest parties and that all honest parties receive valid secret shares.

After the sharing phase, parties must agree on a common set of qualified parties $\mathcal{Q}$ (which includes at least one honest party) that have correctly shared their secrets. This problem is referred to as the agreement on a core set (ACS) problem [1]. Once the ACS phase is completed, it ensures that all honest parties have obtained valid secret shares corresponding to the secrets shared by the parties in $\mathcal{Q}$. Subsequently, the final public key and secret keys are derived from the aggregated secret shares of the parties in $\mathcal{Q}$.

This standard approach incurs high overhead: even with state-of-the art ACSS [7], [8], n parallel instances cost $O(\kappa n^3)$ in communication, and the ACS phase—requiring n parallel Byzantine Agreement (BAs)—also incurs at least $\Omega(\kappa n^3)$ communication. We observe that even with blockchain and random beacons, standalone DKG protocols do not match our efficiency. For instance, consider a strawman solution where we sample a clan of dealers via a random beacon and use state-of-the-art ACSS during the sharing phase. This approach still incurs at least $O(\kappa n_c n^2)$, excluding ACS overhead.

We further observe that non-interactive PVSS-based solutions (see Table 1) are also unable to match the efficiency of our protocol, even with access to random beacons and blockchain. Since these protocols tolerate $f < n/2$ Byzantine faults, electing a clan is not feasible, resulting in no improvement in on-chain storage.

Our Approach

DKG over blockchain using a single consensus instance. To reduce the overhead caused by multiple consensus invocations at various stages of the protocol, we limit the number of consensus instances to a single invocation throughout the protocol execution. Specifically, we send messages via point-to-point channels during the secret sharing phase, avoiding the use of even weaker primitives like RBC. While this approach allows a Byzantine dealer to equivocate and send inconsistent secret shares to different parties, we rely on the non-equivocation guarantees provided by the blockchain to address such inconsistencies at a later point when needed. Additionally, we leverage the blockchain's built-in consensus capabilities to solve the ACS problem. In doing so, our protocol ensures that sufficient information is recorded on-chain to facilitate the derivation of the final DKG public key directly on the blockchain. This approach achieves the goal of using the blockchain's consensus mechanism only once, effectively reducing the round complexity of DKG over blockchain and improving overall efficiency.

We note that the secret sharing phase in our protocol is executed entirely off-chain, with information being posted on the blockchain solely to address the ACS problem. This design choice ensures that our framework remains fully asynchronous, allowing it to operate independently of the network's timing assumptions. Consequently, the exact network model of our protocol is determined by the specific implementation and properties of the underlying blockchain. This flexibility makes our framework adaptable to various blockchains. Notably, this approach enables solving the DKG problem over blockchain in the honest majority setting, even under asynchronous network conditions, as long as blockchain access is available.

Efficient DKG via random sampling. In most standalone DKG protocols, every party in the system acts as a dealer and performs secret sharing. However, it is sufficient to have the contribution of a single honest party to ensure the security of the DKG protocol. Leveraging this observation, we can randomly sample a smaller committee, ensuring that the presence of at least one honest party within the committee is guaranteed with a very high probability. In synchronous network setting, where messages from all honest parties are guaranteed to arrive within a pre-defined time, a significantly smaller committee of dealers can be elected to ensure the presence of at least one honest party [42]. However, in a non-synchronous (e.g., partially synchronous or asynchronous) setting, where up to f_c honest parties may experience arbitrary delays, additional considerations are required. Specifically, to tolerate up to f_c Byzantine failures, we require that $n_c - f_c \geq f_c + 1$, implying that $n_c \geq 2f_c + 1$ parties are necessary. This condition ensures an honest majority in the clan. Therefore, we select a clan of n_c dealers, consisting of at least $2f_c + 1$ parties, to guarantee an honest majority.

The effectiveness of random sampling-based optimization hinges on the system having lower fault tolerance, enabling the selection of a smaller committee with an honest majority, as determined by the hypergeometric probability distribution. Such techniques are particularly suitable for applications like on-chain randomness [63] or consensus mechanisms, where a super-honest majority assumption is typically upheld.

By randomly selecting committee members, significantly smaller committees can be formed while maintaining the honest majority assumption with negligible failure probability. For instance, in a system with $n = 500$ parties and $f = 166$ faulty parties, a committee of only $n_c = 184$ members suffices to ensure an honest majority, with a negligible failure probability of 10^{-9}, as calculated using the hypergeometric probability distribution. Figure 1 illustrates the committee sizes for various system configurations.

In addition to this honest-majority clan, we randomly sample a much smaller "family" of n_a parties, ensuring at least one honest party in the committee with high probability. This family is responsible for posting information to the blockchain. By using such a reduced committee, we minimize interaction with the blockchain while still guaranteeing that at least one honest party posts the necessary information. This approach effectively balances blockchain interaction efficiency with security, minimizing overhead.

We assume the availability of random beacons to enable the fair and unbiased selection of sub-committees. Random beacons [37], which provide publicly verifiable and unbiased randomness, are widely available on most blockchains [62], [64], [73] at no additional cost. Leveraging their utility, as demonstrated in prior works [42], we use random beacons to ensure fairness and unpredictability in the sub-committee sampling process.

Remark. We do not apply committee sampling to scale the consensus protocol itself. A substantial body of work [2], [14], [25], [48] explores this approach, but such protocols require sub-committees with an honest supermajority to run consensus, resulting in large committee sizes unless the system includes over 10,000 participants. Consequently, these techniques are beneficial primarily at very large scales and often provide sub-optimal resilience. Instead, we restrict sub-sampling to selecting the clan of dealers and family for reducing on-chain activity, while treating the blockchain as a black box. This enables performance gains even at moderate system sizes.

4. Preliminaries

We consider a system with n parties $\mathcal{P} := (P_1, \ldots, P_n)$, where up to f parties may be Byzantine. The corruption model is static, meaning the adversary determines which parties are corrupted before the protocol starts. Byzantine parties may behave arbitrarily, while non-faulty parties, referred to as *honest*, strictly adhere to the protocol's specifications.

Our DKG protocol leverages blockchain as a black-box primitive that ensures safety (i.e., parties will not decide on conflicting values) and liveness (i.e., parties will eventually output their input values). The DKG primitive we design is asynchronous, meaning that messages can take an arbitrarily long time to be delivered but are never lost. However, the actual communication model depends on the underlying blockchain.

We employ digital signatures and a public-key infrastructure (PKI) to safeguard against spoofing, replay attacks, and to ensure message authenticity. A message x digitally signed by party P_i using its private key is denoted as $\langle x \rangle_i$, while $\langle x \rangle$ refers to an unsigned message x transmitted over an authenticated channel. Additionally, we represent the hash of an input x by $H(x)$, where H is the hash function.

Setup. Let q be a prime number that is $\mathsf{poly}(\kappa)$ bits long, and let $\mathbb{G}$ be a group of order q such that it is computationally infeasible computing the discrete logarithm in $\mathbb{G}$, except with negligible probability in κ. Denote its scalar field by $\mathbb{Z}_q$. Also, let g denote the generator of $\mathbb{G}$.

Tribe, clan and family. When implementing DKG for the system where super-honest majority assumption is required, the entire system of n parties is referred to as the *tribe*, in which up to $f < \frac{n}{3}$ parties may exhibit Byzantine behavior. A sub-committee within this system, where the honest majority assumption is satisfied, is called a *clan*. In a clan, the number of parties is denoted by n_c, with at most $f_c < \frac{n_c}{2}$ being Byzantine, except with negligible probability. Additionally, a *family* represents a much smaller sub-committee of size n_a that contains at least one honest party i.e., $f_a \leq n_a - 1$ may exhibit Byzantine behavior, except with negligible probability.

4.1. Primitives

Linear erasure and error correcting codes. We utilize standard (n, b) Reed-Solomon (RS) codes [71], which encode b data symbols into n-symbol codewords using the Encode function and decode the codewords to recover the original b data symbols using the Decode function. Additional details on Encode and Decode are provided in Section A.1.

Cryptographic accumulators. A cryptographic accumulator scheme enables the construction of an accumulation value for a set of values via the Eval function and provides a witness for each individual value in the set using the $\mathsf{CreateWit}$ function. Using the accumulation value and a witness, any party can verify whether a specific value is indeed part of the set with the Verify function. More details on these functions are provided in Section A.2.

In this paper, we employ *collision-resistant bilinear accumulators* from Nguyen [67], which provide constant-sized witnesses but rely on the q-SDH assumption [16]. Alternatively, Merkle trees [65] can be used to avoid the q-SDH assumption, with an added communication cost of $O(\log n)$.

Verifiable encryption of committed messages. We use verifiable encryption of committed messages, defined as follows:

Definition 1 (Verifiable Encryption of a Committed Message)**.** *Verifiable encryption (VE) of a committed message involves three entities: a prover $\mathcal{Q}$, a verifier $\mathcal{V}$, and a receiver $\mathcal{R}$. The receiver $\mathcal{R}$ possesses a public-private key pair* $(\mathsf{pk}, \mathsf{sk})$*. Let* cmt *represent a commitment scheme. Given* (v, c, pk)*, the prover $\mathcal{Q}$ seeks to convince the verifier $\mathcal{V}$ that c is a public-key encryption of a message s under the public key* pk*, and that v is a commitment to s, with $\mathcal{Q}$ demonstrating knowledge of s. A verifiable encryption scheme provides the following interfaces:*

- $\mathsf{VE.pp} \leftarrow \mathsf{VE.Setup}(1^\kappa, \mathsf{aux})$*: Generates the scheme parameters* $\mathsf{VE.pp}$*.* $\mathsf{VE.pp}$ *is an implicit input to all other algorithms.*
- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{VE.KGen}(\kappa)$*: Generate public-private key pair* $(\mathsf{pk}, \mathsf{sk})$ *used for share encryption and decryption.*
- $(\vec{E}, \pi) \leftarrow \mathsf{VE.ShareEnc}((s_i)_{i\in[n]}, \mathsf{cmt}, (\mathsf{pk}_i)_{i\in[n]})$*: Given an input of shares* $(s_i)_{i\in[n]}$, cmt *and public keys* $(\mathsf{pk}_i)_{i\in[n]}$*, the algorithm produces a vector of encryptions* $\vec{E} := \mathsf{VE.Enc}(\mathsf{s}_1), \ldots, \mathsf{VE.Enc}(\mathsf{s}_n)$*, and a cryptographic proof of correct sharing* π*.*

- $\{0,1\} \leftarrow \mathsf{VE.Verify}(\mathsf{pk}, \mathsf{cmt}, c, \pi)$*: A deterministic algorithm that verifies the cipher text c with respect to the proof π and commitment* cmt.
- $(s, \pi) \leftarrow \mathsf{VE.Dec}(\mathsf{sk}, c)$*: Given the ciphertext c and a secret key* sk*, the algorithm outputs a decryption of c using* sk.
- $(\mathsf{cmt}, \vec{E}) \leftarrow \mathsf{VE.Aggregate}(\boldsymbol{D})$*: A deterministic algorithm that takes as input a set $\boldsymbol{D}$, consisting of tuples of commitments and encryptions, and outputs an aggregated pair* $(\mathsf{cmt}, \vec{E})$.

Batch interfaces. We also consider a batch interface to generate proofs of correct sharing for a vector of encryptions, as well as a batch interface to verify a vector of encryptions along with their respective proofs of correct sharing.

- $\pi_{VE} \leftarrow \mathsf{VE.BatchProve}(\vec{E}, \mathsf{cmt}, (\mathsf{pk}_i)_{i\in[n]}, \mathsf{wit})$: On input a vector of encryptions $\vec{E}$, their corresponding commitments cmt, and witness wit, the algorithm outputs a NIZK proof π_{VE} that satisfies the verification requirements of $\mathrm{VE.BatchVerify}$.
- $\{0,1\} \leftarrow \mathsf{VE.BatchVerify}(\mathsf{cmt}, I, \vec{E}_I, \pi_{VE})$: The algorithm outputs 1 if π_{VE} is a valid proof demonstrating that, for each $i \in I$, there exists (α_i, π_i) such that $(\alpha_i, \pi_i) = \mathsf{VE.Dec}(\mathsf{sk}_i, c_i)$.

State machine replication aka blockchain. We leverage existing state machine replication (SMR) protocols or blockchain as a black-box primitive to realize our DKG protocol. The blockchain protocol, executed by a group of parties, processes input client requests and outputs a linearizable log, ensuring a consistent view of the log, comparable to that of a single non-faulty party. It offers the following guarantees, with a formal definition provided for the partially synchronous model in Figure 9 (in Appendix).

- **Safety.** Honest parties do not commit different values at the same log position.
- **Liveness.** Each client request is eventually committed by all honest parties.

4.2. DKG Definition

A Distributed Key Generation (DKG) protocol for n parties $(P_1, \ldots, P_n)$ produces private outputs, referred to as *shares*, denoted by $(x_1, \ldots, x_n)$, which collectively represent the shares of a random value $x \in \mathbb{Z}_q$. The protocol also outputs a public value $y = g^x$, where g is a randomly chosen generator of a group $\mathbb{G}$ of order q. Furthermore, each party obtains the threshold public keys corresponding to all parties, specifically g^{x_i} for all $i \in [n]$.

In a DKG protocol, the degree ℓ of the polynomial used to share the secret is referred to as the *reconstruction threshold*. The threshold ℓ determines the minimum number of parties required to reconstruct the secret and can range between f and $n-f-1$. A DKG protocol is classified as *low-threshold* when $\ell = f$ and as *high-threshold* when $\ell > f$.

In this work, we focus on a biasable DKG protocol where the adversary is capable of biasing the final public key y. The formal functionality is outlined in Figure 2. Our functionality extends the biasable DKG functionality initially proposed by Cascudo et al. [22] for the synchronous network setting. We adapt their framework to operate in non-synchronous settings and further extend it to support a dual-threshold DKG configuration, wherein the reconstruction threshold ℓ can exceed the corruption threshold f.

A key feature of our functionality is its allowance for the adversary to bias not only the final public key but also the individual threshold public keys of a specified number of parties. In contrast, the DKG functionality of [58] permits biasing the final public key but not the individual public keys in their "Strong mode", which assumes a majority of honest parties. However, their non-interactive DKG protocol enables a rushing adversary to bias individual public keys by observing the PVSS vectors submitted by honest parties and subsequently selecting their own PVSS vectors accordingly. This adversarial capability is not accurately reflected in their DKG functionality.

Similarly, the biasable DKG functionality (Fig. 8 in [59]) proposed by Katz captures the adversary's ability to bias the final public key but fails to account for the possibility of biasing the individual threshold public keys of participating parties. Our work addresses these limitations, providing a more comprehensive formalization of the adversary's capabilities in biasable DKG protocols.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ denote the hash size, $\kappa_a = \kappa_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator and $\kappa_v = \kappa_v(\lambda)$ denote the size of secret share and witness of a secret. Further, let $\kappa = \mathsf{max}(\kappa_h, \kappa_a, \kappa_v)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_v) = \Theta(\kappa_a) = \Theta(\lambda)$. Throughout the paper, we can use the same parameter κ to denote the hash size, signature size, accumulator size and secret share size for convenience.

5. Efficient Message Propagation

This section introduces two efficient propagation primitives–Deliver and CS-Deliver—for disseminating long messages with low communication overhead.

Deliver function. We introduce a Deliver function (see Figure 3) that enables an honest party to efficiently disseminate long messages using erasure coding and cryptographic accumulators. Prior to invoking the Deliver function, a clan of n_c parties is assumed to have been selected. This function is adapted from RandPiper [13], extended to the asynchronous setting and optimized for the efficient propagation of long messages within the clan $\mathcal{C}$.

The Deliver function takes as input a long message m and its corresponding accumulation value z, which represents m. It is assumed that all parties in the clan or tribe have already agreed on the accumulation value z prior to invoking the function. The function operates by encoding the message m into n_c code words $[s_1, \ldots, s_{n_c}]$ using (n_c, f_c+1) RS codes and generating corresponding witnesses $[w_1, \ldots, w_{n_c}]$ for each code word. Each code word and its witness pair (s_j, w_j)

Let $\mathcal{C}$ denote an arbitrary set of up to f parties, controlled by the ideal-world adversary $\mathcal{S}$, with $\mathcal{C} \subset [n]$. Let $\ell \in [f, n-f-1]$ represent the reconstruction threshold. Additionally, let $\mathbb{G}$ be an elliptic curve group of order q with generator g and scalar field $\mathbb{Z}_q$. Initialize an empty set $\mathcal{V} = \emptyset$.

1) Upon receiving (Gen, sid, P_i) from an honest party P_i, add i to $\mathcal{V}$ and forward (Gen, sid, P_i) to $\mathcal{S}$. Wait until $|\mathcal{V}| > 1$.
2) Sample a random polynomial $p(.)$ of degree ℓ. Set $y = g^{p(0)}$ and $y'_i = g^{x'_i}$ where $x'_i = p(i)\ \forall i \in [n]$.
3) Send $(\mathsf{KEYS}, sid, \{y_j\}_{j\in[n]}, y, \{x_j\}_{j\in\mathcal{C}})$ to $\mathcal{S}$.
4) Upon receiving (BIAS, sid, p') from $\mathcal{S}$, where p' is a polynomial of degree ℓ, update $y' = y \cdot g^{p'(0)}$, $y'_i = y_i \cdot g^{p'(i)}$ and $x'_i = x_i + p'(i)\ \forall i \in [n]$.
5) Send $(\mathsf{KEYS}, x'_i, \{y'_i\}_{i\in[n]}, y')$ to party $P_i\ \forall i \in [n]$.

Figure 2: **Biasable DKG Ideal Functionality $\mathcal{F}_{\mathsf{BDKG}}$**

$\mathsf{Deliver}(m, z)$:

- Partition input m into $f_c + 1$ blocks. Encode the b blocks into n_c code words $[s_1, \ldots, s_{n_c}]$ using Encode function. Add an index j to each code word s_j to obtain $\mathbf{D} = [(1, s_1), \ldots, (n, s_{n_c})]$. Compute accumulation value $z' = \mathsf{Eval}(a_k, \mathbf{D})$. If $z \neq z'$, abort. Otherwise, Compute witness w_j for each element $(j, s_j) \in \mathbf{D}$ using $\mathsf{CreateWit}$ function and send $\langle \mathsf{codeword}, s_j, w_j \rangle$ to j^{th} party in $\mathcal{C}\ \forall j \in [n_c]$.
- If the j^{th} party in $\mathcal{C}$ receives a valid code word $\langle \mathsf{codeword}, s_j, w_j \rangle$ such that $\mathsf{Verify}(a_k, z, w_j, (j, s_j)) = \mathsf{true}$ and has not yet sent this code word, forward the code word to all the parties.
- Upon receiving $f_c + 1$ valid code words for the accumulation value z it received, decode m using Decode function.

Figure 3: **Deliver function**

$\mathsf{CS\text{-}Deliver}(m)$:

- Partition input m into $2f + 1$ blocks. Encode the $2f + 1$ blocks into n code words $[s_1, \ldots, s_n]$ using Encode function. Add an index j to each code word s_j to obtain $\mathbf{D} = [(1, s_1), \ldots, (n, s_n)]$. Compute accumulation value $z = \mathsf{Eval}(a_k, \mathbf{D})$. Compute witness w_j for each element $(j, s_j) \in \mathbf{D}$ using $\mathsf{CreateWit}$ function and send $\langle \mathsf{codeword}, s_j, w_j, z \rangle_i$ to party $P_j\ \forall j \in [n]$.
- If party P_j receives $f_c + 1$ identical valid code word $\langle \mathsf{codeword}, s_j, w_j, z \rangle_*$ such that $\mathsf{Verify}(a_k, z, w_j, (j, s_j)) = \mathsf{true}$ and has not yet sent this code word, forward the code word to all the parties.
- Upon receiving $2f + 1$ valid code words for the accumulation value z it received, decode m using Decode function.

Figure 4: **Consortium Sender Deliver function**

is then sent to the j^{th} party in the clan, with these pairs being transmitted unsigned. Since all parties have already agreed on the accumulation value z, they can independently verify the validity of each code word using its corresponding witness. This verification is robust against attempts by Byzantine parties to send multiple or invalid code words, eliminating the need for signing these messages.

Upon receiving the pair (s_j, w_j) for an accumulation value z, the j^{th} party verifies that (j, s_j) corresponds to the j^{th} element of the set $\mathbf{D}$ with the given accumulation value z and the provided witness w_j. Once verified, the j^{th} party forwards the first valid code word and witness it received to other members of the clan. When an honest party collects $f_c + 1$ valid code words corresponding to the accumulation value z, it can decode the original message m.

Invoking the Deliver function on a message of size ℓ bits incurs a communication cost of $O(n_c\ell + (\kappa + w)n_c^2)$, where κ represents the size of the accumulator and w is the size of the accumulator witness. When using bilinear accumulators, the size of the witness is $O(\kappa)$, while with Merkle trees, it is $O(\kappa \log n_c)$. Consequently, the total communication complexity to propagate a single message of size ℓ is $O(n_c\ell + \kappa n_c^2)$ bits when relying on bilinear accumulators, or $O(n_c\ell + \kappa n_c^2 \log n_c)$ bits when using Merkle trees.

Consortium sender deliver function. We also introduce the consortium sender Deliver function (refer to Figure 4), designed to enable a consortium of parties within the clan to efficiently propagate a common long message to all parties in the tribe. This function leverages erasure coding and cryptographic accumulators to optimize message dissemination. Inspired by Dragon-DKG [41], it allows the parties in the clan to collectively and efficiently disseminate a pre-agreed message m to the parties in the tribe.

The CS-Deliver function is invoked by the parties in the clan with a common long message m as input. It encodes the message m into n code words $[s_1, \ldots, s_n]$ using $(n, 2f+1)$ Reed-Solomon (RS) codes and generates corresponding witnesses $[w_1, \ldots, w_n]$ for each code word. Each code word and its witness pair (s_j, w_j) is then transmitted to the respective party P_j along with the accumulation value z. Unlike the Deliver function, the message containing the code word, witness, and accumulation value is signed to ensure authenticity.

Each party P_j then awaits receipt of identical valid codeword, witness, and accumulation value (s_j, w_j, z) from at least $f_c + 1$ distinct parties. This ensures that the message m corresponding to the accumulation value z is indeed the value agreed upon by the parties in the clan. Once this condition is satisfied, P_j forwards the tuple (s_j, w_j, z) only once to all other parties in the tribe. Upon collecting at least $2f + 1$ valid codewords associated with the accumulation value z, any honest party can decode and reconstruct the original message.

Invoking the CS-Deliver function on a message of size ℓ bits incurs a communication cost of $O(n\ell + (\kappa + w)n^2)$, where κ represents the size of the accumulator and w is the size of the accumulator witness. When using bilinear accumulators, the size of the witness is $O(\kappa)$, while with Merkle trees, it is $O(\kappa \log n_c)$. Consequently, the total communication complexity to propagate a single message of size ℓ is $O(n\ell + \kappa n^2)$ bits when relying on bilinear accumulators, or $O(n\ell + \kappa n^2 \log n)$ bits when using Merkle trees.

6. Efficient DKG for Use over Blockchain

In this section, we present an efficient DKG protocol designed for blockchain settings. We first describe the protocol under the super-honest majority assumption, and later extend it to the more relaxed honest-majority case.

Protocol details. We present our efficient DKG protocol in Figure 5. While the protocol is designed to support any threshold within the range $f \leq \ell \leq 2f$, we specifically focus on the highest threshold, $\ell = 2f$, as it is particularly well-suited for applications such as consensus protocols and on-chain randomness. As previously discussed, we assume the parties have access to a randomness beacon RB to select a smaller clan of dealers. The beacon produces an output (run on a predefined timestamp) and a verifiable proof along with it that attests to the correct output computation. The output is used to select the smaller clan of dealers. We rely on the unpredictability (refer to Appendix 2) of the RB output to ensure that the clan members are randomly selected.

It is important to note that the contribution of a single honest party is sufficient to ensure the security of the DKG. In a partially synchronous or asynchronous network, $n_c \geq 2f_c + 1$ parties are necessary to ensure this property (as discussed before). Therefore, we select a clan of n_c dealers (denoted by $\mathcal{C}$), consisting of at least $2f_c+1$ parties, to ensure an honest majority within the clan with high probability. We also elect a family $\mathcal{F}$ consisting of n_a parties, ensuring it includes at least one honest party with high probability. Note that the family $\mathcal{F}$ is significantly smaller in size. To minimize interaction with the blockchain, our protocol allows the parties in $\mathcal{F}$ to post information onto the blockchain.

To reduce communication overhead, the responsibility of sharing secrets is delegated solely to the clan of dealers. For sharing secrets, we adopt the recent VSS protocol proposed by Das et al. [31], which is known for its computational efficiency. In their scheme, each dealer P_i selects a random degree $2f$ polynomial $p_i(x)$ and sends individual secret shares $p_i(j)$ to each party P_j, along with a linear-sized discrete log commitment cmt_i which are commitment to the coefficients of the secret-sharing polynomial $p_i(x)$. Each party then verifies the correctness of the commitment via SCRAPE's low degree check [21] and checks its received secret share against the commitment cmt_i. If the verification succeeds, the party signs cmt_i and sends the signature back to the dealer. Once the dealer collects a quorum (i.e., a set of $2f+1$) of valid signatures, denoted as σ_i, the dealers use verifiable encryptions to distribute the secret shares for parties with missing signatures, accompanied by σ_i via a reliable broadcast (RBC) primitive. The RBC primitive is used to ensure the dealer sends consistent value to all the parties. Once the RBC terminates, the parties accept the dealer's secret sharing if σ_i contains a quorum of valid signatures and the encrypted secret shares for the missing parties are valid. If either condition is not satisfied, the dealer's secret sharing is rejected. Notably, in their scheme, each party is required to verify only f encryptions, which is critical for reducing computational overhead.

While the above secret sharing approach minimizes the computational overhead, it still incurs higher communication when sending the linear sized commitment vector to all the parties which incurs at least $O(\kappa n^2)$ communication per dealer. Moreover, each RBC invocation requires $\Omega(n^2)$ communication [36]. As our DKG protocol involves at least n_c dealers, their approach would incur at least $O(\kappa n_c n^2)$ in communication.

To reduce communication overhead, we propose an improved VSS protocol based on Das et al. [31], which reduces communication in their approach. In our protocol, the dealer P_i sends only the individual secret share $p_i(j)$ directly to each party P_j without transmitting any commitments to verify the secret share. Each party P_j computes the discrete log commitment of their share, $g^{p_i(j)}$ (denoted as $\mathsf{cmt}_{i,j}$), signs it, and sends the signature back to the dealer P_i. The dealer P_i collects a quorum of signed commitments for the individual secret shares (denoted as σ_i) and computes the entire commitment vector cmt_i, which contains commitments to the evaluations of the secret-sharing polynomial $p_i(.)$. For parties missing valid signatures, identified as I_i, the dealer generates verifiable encryptions of the corresponding secret shares, denoted by $\vec{E}_{I_i}$ and its corresponding proof π_{I_i}.

The dealer then transmits the commitment vector cmt_i, the quorum of signatures σ_i, the verifiable encryptions $\vec{E}_{I_i}$ and its corresponding proof π_{I_i} only to the parties in $\mathcal{C}$ via point-to-point channels, avoiding the use of reliable broadcast (RBC). It is worth noting that an honest sender always sends consistent value to all parties in the clan, whereas a Byzantine sender may send equivocating values. We address any such equivocation later using the blockchain, which inherently provides non-equivocation guarantees.

In our protocol, only the clan $\mathcal{C}$ of parties is responsible for verifying the commitments, signatures, and the verifiable encryptions. This approach avoids the need for all parties to perform these verifications, reducing the overall computational and communication overhead. Upon receiving the commitment vector cmt_i, the quorum of signatures σ_i, and the verifiable encryptions $\vec{E}_{I_i}$, each party P_j in $\mathcal{C}$ performs several verification steps. First, it applies SCRAPE's low-degree check to validate the correctness of the commitment. Next, it verifies the signature $\sigma_{i,j}$ associated with the signed commitment $\mathsf{cmt}_{i,j}$. Finally, it checks the encrypted secret shares corresponding to any missing signatures. If all these checks are successful, party P_j signs a Vote message for the first element of cmt_i i.e., $g^{p_i(0)}$ and the accumulation value of $(\mathsf{cmt}_i, \vec{E}_{I_i})$, denoted as z_i. The signed Vote message is sent only to the parties in the family $\mathcal{F}$.

Receiving $f_c + 1$ Vote messages for $(\mathsf{cmt}_i, \vec{E}_{I_i})$ ensures that at least one honest party has verified the correctness of cmt_i, σ_i, and $\vec{E}_{I_i}$. Consequently, parties in $\mathcal{F}$ wait to receive at least $f_c + 1$ votes for $(\mathsf{cmt}_i, \vec{E}_{I_i})$. Upon receiving $f_c + 1$ Vote messages for the same $\mathsf{cmt}_i[0]$ and the accumulation value z_i, party $P_k \in \mathcal{F}$ forms a quorum certificate for $(z_i, \mathsf{cmt}_i[0])$, denoted as $\mathsf{QC}(z_i, \mathsf{cmt}_i[0])$. Party P_k then updates its meta DKG instance MDKG_k by adding z_i, appends $\mathsf{cmt}_i[0]$ to its commitment list CList_k, and includes $\mathsf{QC}(z_i, \mathsf{cmt}_i[0])$ in its list of quorum certificates LQC_k. The presence of $f_c + 1$ Vote messages for the same z_i ensures data availability for $(\mathsf{cmt}_i, \vec{E}_{I_i})$, as at least one honest party must have received $(\mathsf{cmt}_i, \vec{E}_{I_i})$, enabling its

Each party $P_i \in \mathcal{P}$ executes $\mathsf{VE.Setup}$ to generate the public parameters $\mathsf{VE.pp}$, which are agreed upon by all parties. Next, each party P_i generates a key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{VE.KGen}(\kappa)$, with all parties agreeing on each other's public keys. Using a random beacon RB, the parties select a clan $\mathcal{C}$ of n_c parties and a family $\mathcal{F}$ of n_a parties from the entire tribe. Note that $n_a << n_c << n$. Additionally, each party $P_k \in \mathcal{F}$ initializes $\mathsf{MDKG}_k \leftarrow []$, $\mathsf{CList}_k \leftarrow []$ and $\mathsf{LQC}_k \leftarrow []$.

```
1) Each party $P_i \in \mathcal{C}$ (as a dealer) performs the following:
      Sample a $2f$ degree random polynomial $p_i(x)$ such that $p_i(0) = s_i$
      send $\langle \mathsf{Share}, p_i(j)\rangle_i$ to party $P_j$ $\forall j \in \{1, ..., n\}$
2) Upon receiving $\langle \mathsf{Share}, p_j(i)\rangle_j$ from the dealer $P_j$,
      $\mathsf{cmt}_{j,i} \leftarrow g^{p_j(i)}$
      Let $\sigma_{j,i} := \mathsf{Sign}(\mathsf{sk}_i, \mathsf{cmt}_{j,i})$
      send $\sigma_{j,i}$ to dealer $P_j$
3) Upon receiving a set $\sigma_i$ of $2f+1$ valid signatures, each dealer $P_i \in \mathcal{C}$ performs the following:
      $\mathsf{cmt}_i \leftarrow [g^{p_i(0)}, g^{p_i(1)}, g^{p_i(2)}, \ldots, g^{p_i(n)}]$
      Let $I_i$ be indices of parties with missing valid signatures.
      $(\vec{E}_{I_i}, \mathsf{wit}_{I_i}) \leftarrow \mathsf{VE.ShareEnc}(p_i(j)_{j \in I_i}, \mathsf{cmt}_i, (\mathsf{pk}_j)_{j \in I_i})$
      $\pi_{I_i} \leftarrow \mathsf{VE.BatchProve}(\vec{E}_{I_i}, \mathsf{cmt}_i, (\mathsf{pk}_j)_{j \in I_i}, \mathsf{wit}_{I_i})$
      send $\langle \mathsf{Propose}, \mathsf{cmt}_i, I_i, \sigma_i, \vec{E}_{I_i}, \pi_{I_i}\rangle_i$ to each party $P_j \in \mathcal{C}$
4) Upon receiving $\langle \mathsf{Propose}, \mathsf{cmt}_j, I_j, \sigma_j, \vec{E}_{I_j}, \pi_{I_j}\rangle_j$, each party $P_i \in \mathcal{C}$ performs the following:
      Check $\mathsf{PC.DegCheck}(\mathsf{cmt}_j, 2f)$
      Check if each $\sigma_{j,k} \in \sigma_i$ is valid signature on $\mathsf{cmt}_{j,k}$ and $|\sigma_j| \geq 2f+1$
      Check that each $I_j$ includes indices of all parties with missing valid signatures
      Check if $\mathsf{VE.BatchVerify}(\mathsf{cmt}_j, I_j, \vec{E}_{I_j}, \pi_{I_j})$
      **if** all the check pass **then**
         send $\langle \mathsf{Vote}, z_i, \mathsf{cmt}_j[0]\rangle_i$ to each party $P_k \in \mathcal{F}$     $\triangleright$ $z_i$ is the accumulation value of $(\mathsf{cmt}_i, \vec{E}_{I_j})$
5) For each party $P_k \in \mathcal{F}$, upon receiving $f_c + 1$ distinct votes for $(z_i, \mathsf{cmt}_i[0])$:
      $\mathsf{MDKG}_k \leftarrow \mathsf{MDKG}_k \cup \{z_i\}$
      $\mathsf{CList}_k \leftarrow \mathsf{CList}_k \cup \{\mathsf{cmt}_i[0]\}$
      $\mathsf{LQC}_k \leftarrow \mathsf{LQC}_k || \mathsf{QC}(z_i, \mathsf{cmt}_i[0])$     $\triangleright$ $\mathsf{QC}(z_i, \mathsf{cmt}_i[0])$ denotes $f_c + 1$ votes on $(z_i, \mathsf{cmt}_i[0])$
      **if** $|\mathsf{MDKG}_k| \geq f_c + 1$ **then**
         compute $g^x$ from all elements in $\mathsf{CList}_k$
         Multicast $\langle \mathsf{MetaDKG}, g^x, \mathsf{MDKG}_k, \mathsf{CList}_k, \mathsf{LQC}_k\rangle_k$ to party $P_j \in \mathcal{C}$.
6) Upon receiving the first $\langle \mathsf{MetaDKG}, g^x, \mathsf{MDKG}_k, \mathsf{CList}_k, \mathsf{LQC}_k\rangle_k$ from party $P_k$, each party $P_i \in \mathcal{C}$ performs the
   following:
      Verify that $\mathsf{MDKG}_k$ and $\mathsf{CList}_k$ contain $f_c + 1$ elements from distinct parties, each $(z_i, \mathsf{cmt}_i[0])$ paired with a valid
      QC
      Check if $g^x$ is computed based on $\mathsf{CList}_k$
      **if** the above check pass **then**
         send $\langle \mathsf{Ack}, (g^x, z_{mk})\rangle_i$ to party $P_k$ in $\mathcal{F}$     $\triangleright$ $z_{mk}$ is the accumulation value of $\mathsf{MDKG}_k$
7) For each party $P_k \in \mathcal{F}$, upon receiving $f_c + 1$ distinct $\mathsf{Ack}$ on $z_{mk}$, send $(Send, \mathsf{sid}, (g^x, z_{mk}, \mathsf{AC}(g^x, z_{mk})))$ to $\mathcal{F}_{\mathsf{BC}}$.
8) Each party runs $\mathcal{F}_{\mathsf{BC}}$ on $(Get, \mathsf{sid})$ to read the SMR as $\mathsf{TO}_i^{\mathsf{sid}}$. Let $(g^x, z_{mk}, \mathsf{AC}(z_{mk}))$ be the first valid output from
   $\mathcal{F}_{\mathsf{BC}}$.
   Invoke $\mathsf{Deliver}(\mathsf{MDKG}_k, z_{mk})$ to propagate $\mathsf{MDKG}_k$.
9) For each party $P_i \in \mathcal{C}$ when $\mathsf{MDKG}_k$ has been delivered, invoke $\mathsf{Deliver}((\mathsf{cmt}_j, \vec{E}_{I_j}), z_j)$ for all $z_j \in \mathsf{MDKG}_k$.
10) Upon receiving all $(\mathsf{cmt}_j, \vec{E}_{I_j})$ such that $z_j \in \mathsf{MDKG}_k$,
      $(\mathsf{cmt}, \vec{E}) \leftarrow \mathsf{VE.Aggregate}(\{(\mathsf{cmt}_j, \vec{E}_{I_j}) | z_j \in \mathsf{MDKG}_k\})$
      $\mathcal{Q} \leftarrow [j \mid z_j \in \mathsf{MDKG}_k]$ and $x_i \leftarrow \mathsf{VE.Dec}(\mathsf{sk}, \vec{E}[i]) + \sum_{j \in \mathcal{Q}} p_j(i)$
      send $\langle \mathsf{EncShare}, \vec{E}[j]\rangle_i$ to party $P_j \in \mathcal{P} \setminus \mathcal{C}$ and invoke $\mathsf{CS\text{-}Deliver}((\mathsf{cmt}, \mathcal{Q}))$
11) Upon receiving $\langle \mathsf{EncShare}, \vec{E}[i]\rangle_*$ from $f_c + 1$ parties and $\mathcal{Q}$ list and perform $x_i \leftarrow \mathsf{VE.Dec}(\mathsf{sk}, \vec{E}[i]) + \sum_{j \in \mathcal{Q}} p_j(i)$.
```

Figure 5: **Efficient DKG Π_{DKG} for use over Blockchain**

retrieval when required. Once MDKG_k (and CList_k) contains f_c+1 elements, it guarantees that at least one honest party's contribution is included in the meta DKG instance MDKG_k with high probability.

In principle, the meta DKG instance MDKG_k, along with CList_k and the aggregated public key g^x (computed from CList_k), could be directly posted to the blockchain, with the first instance output by the blockchain being used to derive the DKG secret and public keys. However, storing each LQC, which can grow quadratically in size, on the blockchain is highly inefficient. Moreover, limiting the number of postings to the blockchain by parties in the family $\mathcal{F}$ is desirable. To address these challenges, parties in $\mathcal{F}$ first compute the public key g^x from CList_k by aggregating its elements. They then distribute MDKG_k, CList_k, and the computed g^x to all parties in the clan $\mathcal{C}$, along with the necessary quorum certificates (LQCs) for verification. This approach ensures communication and verification efficiency without overloading the blockchain.

Upon receiving MDKG_k, CList_k, and the corresponding g^x along with the LQCs that certify each element of MDKG_k (and CList_k), and verifying that g^x is the aggregate of all elements in CList_k, each party P_j in the clan sends an Ack message for (g^x, z_{mk}) to party P_k, where z_{mk} is the accumulation value of the meta DKG. It is important to note that an honest party sends at most one Ack message per party in the family $\mathcal{F}$. When party P_k receives at least f_c+1 Ack on (g^x, z_{mk}) (denoted by $\mathsf{AC}(g^x, z_{mk})$), it posts $(g^x, z_{mk}, \mathsf{AC}(g^x, z_{mk}))$ to the blockchain for agreement.

Let (g^x, z_{mk}) denote the first valid output from the blockchain. Observe that z_{mk} represents the accumulation value of the meta DKG instance MDKG_k created by party $P_k \in \mathcal{F}$. When P_k is honest, all parties in $\mathcal{C}$ receive MDKG_k. However, if P_k is Byzantine, it is possible that only a single honest party receives MDKG_k. To efficiently propagate MDKG_k within $\mathcal{C}$, we leverage the $\mathsf{Deliver}$ function. As previously noted, since z_{mk} is already known to all parties, the $\mathsf{Deliver}$ function does not require signatures, enabling efficient dissemination of MDKG_k.

Once MDKG_k has been propagated, all honest parties in $\mathcal{C}$ obtain the z_j values included in MDKG_k. Importantly, at least one honest party must have received $(\mathsf{cmt}_j, \vec{E}_{I_j})$. We further use the $\mathsf{Deliver}$ function to propagate $(\mathsf{cmt}_j, \vec{E}_{I_j})$ efficiently for all $z_j \in \mathsf{MDKG}_k$.

Upon receiving $(\mathsf{cmt}_j, \vec{E}_{I_j})$ for all $z_j \in \mathsf{MDKG}_k$, the parties in $\mathcal{C}$ aggregate these inputs to compute the common pair $(\mathsf{cmt}, \vec{E})$. They first identify the list of dealers in MDKG_k, denoted as $\mathcal{Q}$. Here, $\vec{E}$ contains the aggregated encryption vector for the missing encryptions associated with dealers in $\mathcal{Q}$. The honest parties in the tribe would have already received the remaining secret shares directly from these dealers. Each party P_i in $\mathcal{C}$ computes its final secret key by combining the secret shares sent by parties in $\mathcal{Q}$ with the decryption of $\vec{E}[i]$.

The parties in $\mathcal{C}$ propagate $(\mathsf{cmt}, \mathcal{Q})$ using the $\mathsf{CS\text{-}Deliver}$ primitive. This primitive ensures that a pre-agreed message is delivered from the clan to all parties in the tribe. Additionally, the parties in $\mathcal{C}$ send the corresponding $\vec{E}[i]$ to each party $P_i \in \mathcal{P}$. Upon receiving $\mathcal{Q}$, each party P_i computes its partial secret key x_i' using the secret shares sent by parties in $\mathcal{Q}$ earlier in the protocol. Additionally, upon receiving at least f_c+1 consistent copies of $\vec{E}[i]$, each party P_i decrypts $\vec{E}[i]$ and combines it with x_i' to compute its final secret key x_i. The threshold public keys for all parties are derived directly from the aggregated commitment.

6.1. Efficiency Analysis

Communication complexity. At the start of the protocol, each party $P_i \in \mathcal{C}$ (as a dealer) sends $O(\kappa)$-sized secret share to all the parties in the tribe, resulting in $O(\kappa n_c n)$ bits of communication. In the next round, all parties in the tribe send a $O(\kappa)$-sized signature back to the dealers in the clan, also incurring $O(\kappa n_c n)$ in communication. Then, each dealer $P_i \in \mathcal{C}$ sends $O(\kappa n)$-sized message containing the list of signatures, commitment, indices, encryptions, and proof of correct sharing to parties in $\mathcal{C}$, incurring $O(\kappa n_c^2 n)$ bits in communication. Subsequently, the parties in the clan send $O(\kappa)$-sized Vote message and signature for each dealer to the parties in $\mathcal{F}$ resulting in a communication cost of $O(\kappa n_a n_c^2)$.

Next, parties in $\mathcal{F}$ collect Vote messages and form at least f_c+1 QCs. The size of QC depends on the signature scheme. For instance, its size is $O(\kappa + n_c)$ when using BLS multi-signatures and $O(\kappa)$ or $O(\kappa \log n_c)$ when using threshold signatures. Threshold signatures can be instantiated without needing DKG, resulting in a signature of size $O(\kappa \log n_c)$ with a transparent setup [10] or $O(\kappa)$ with a CRS setup [30], though these threshold signatures remain largely theoretical. Consequently, the size of a meta DKG and CList is $O(\kappa n_c)$ and the size of list of QCs is $O(\kappa n_c + n_c^2)$ when using BLS multi-signatures, or $O(\kappa n_c)$ when using threshold signatures. When the parties in $\mathcal{F}$ send their meta DKG and CList along with the list of QCs to all parties in the clan, the communication complexity is $O(\kappa n_a n_c^2 + n_a n_c^3)$ with multi-signatures, or $O(\kappa n_a n_c^2)$ with threshold signatures.

Next, the parties in $\mathcal{C}$ send the Ack messages back to the parties in $\mathcal{F}$, resulting in a communication cost of $O(\kappa n_a n_c)$. Finally, the parties in $\mathcal{F}$ submit the cryptographic accumulator of the meta DKG and the corresponding Ack certificate, with a size of $O(\kappa + n_c)$ with multi-signature or $O(\kappa)$ with threshold signature, to the SMR protocol. Since there are n_a parties in $\mathcal{F}$, up to $O(\kappa n_a + n_a n_c)$ or $O(\kappa n_a)$ bits may be submitted to the SMR protocol.

We analyze the communication cost of the consensus process separately, as it is a necessary step for all implementations to submit the final public key onto the blockchain. When the SMR protocol outputs the cryptographic accumulator of MDKG_k, the parties in $\mathcal{C}$ propagate MDKG_k using the $\mathsf{Deliver}$ primitive. Since the size of the meta DKG is $O(\kappa n_c)$, by Claim 2, this propagation incurs $O(\kappa n_c^2)$ bits of communication. Additionally, the parties in $\mathcal{C}$ propagate the encryption vectors and commitments within MDKG_k. Each encryption and commitment is of size $O(\kappa n)$, and propagating n_c

encryptions via the Deliver primitive, as per Claim 2, incurs $O(\kappa n_c^2 n)$ bits of communication. Subsequently, the parties in $\mathcal{C}$ send the aggregated encryption to all parties in the tribe, incurring $O(\kappa n_c n)$ communication cost. By Claim 4, delivering the aggregated commitment, which is $O(\kappa n)$ in size, to all parties in the tribe incurs an additional $O(\kappa n^2)$. Consequently, the total off-chain communication cost of the protocol (excluding consensus) is $O(\kappa n_c^2 n + n_a n_c^3 + \kappa n^2)$ when using multi-signatures, and $O(\kappa n_c^2 n + \kappa n^2)$ when using threshold signatures.

The communication cost for consensus is $\Omega(n\ell + n^2)$ for ℓ bit input [36], [45], with existing constructions achieving $O(n\ell + \kappa n^2)$ [74]. Given an SMR input of size $O(\kappa n_a + n_a n_c)$ with multi-signature or $O(\kappa n_a)$ with threshold signatures, the consensus cost becomes $O(n_a n_c n + \kappa n^2)$ with multi-signatures or $O(\kappa n^2)$ with threshold signatures. We stress again that this communication cost is inherent in all applications that aim to use threshold cryptography over a blockchain.

Round complexity. In the best case, when all dealers are honest, all parties in the clan will receive the meta DKG along with the associated encryptions and commitments, allowing the protocol to complete in 8 rounds plus the consensus latency to commit the first valid meta DKG, which requires at least 3 rounds in the good-case. In contrast, if the dealers are Byzantine, parties in the clan will need to first download the meta DKG and then the associated encryptions, which requires an additional 4 rounds.

Computation complexity. Observe that only parties in $\mathcal{C}$ are responsible for verifying the encryptions and list of signatures which amounts to $O(n_c n)$ computation per party. The parties in the $\mathcal{F}$ verify only list of QC which incurs $O(n_c^2)$ computation. The rest of the parties perform $O(n_c)$ computation.

Statistical security analysis. When a single clan of n_c parties is randomly selected from a system of n total parties, of which f are Byzantine, the probability of forming a dishonest majority within the clan is given by the hypergeometric cumulative distribution function (CDF). This function calculates the likelihood of selecting more than half of the n_c parties as Byzantine. Specifically, the probability $\Pr(\text{dishonest majority})$ is:

$$\Pr(\text{dishonest majority}) = \sum_{k=\lceil \frac{n_c}{2} \rceil}^{n_c} \frac{\binom{f}{k}\binom{n-f}{n_c-k}}{\binom{n}{n_c}} \tag{1}$$

The value of n_c must be chosen sufficiently large relative to n and f so that the probability of forming a dishonest majority clan is below the desired security threshold μ. Specifically,

$$\Pr(\text{dishonest majority}) \leq 2^{-\mu} \tag{2}$$

Similarly, probability that a randomly selected committee contains no honest party is determined using hypergeometric distribution. Specifically, if there are n parties, of which f are Byzantine, and a committee of size n_a is selected, the probability of selecting a committee with no honest parties is:

$$\Pr(\text{No honest party}) = \frac{\binom{f}{n_a}}{\binom{n}{n_a}} \tag{3}$$

The value of n_a must be selected in proportion to n and f to ensure that the probability of forming a family without any honest party remains below the desired security threshold μ.

$$\Pr(\text{No honest party}) \leq 2^{-\mu} \tag{4}$$

6.2. DKG in the Honest Majority Setting

In the above protocol, we focused on DKG for applications like consensus and on-chain randomness, where the DKG setup involves the parties participating in consensus. For such settings, $f < n/3$ is required in partially synchronous or asynchronous network settings [38].

When setting up DKG for an external system with access to a blockchain, we can tolerate up to $f < n/2$ Byzantine faults within the external system. In such scenarios, our protocol from Figure 5 can be adapted for use over the blockchain with the following modifications. Since the system operates under an honest majority assumption, sampling a clan is unnecessary, as it would result in a subcommittee nearly the size of the original system. Instead, a family of parties can be elected using random beacons to minimize on-chain postings. The required adjustments include setting the degree of the secret sharing polynomial to $f+1$. Additionally, the CS-Deliver function for propagating aggregated commitments and sending individual aggregate encryptions to the parties becomes unnecessary, as all parties are part of the committee.

Efficiency analysis. The protocol incurs communication complexity of $O(\kappa n^3 + n_a n^3)$, on-chain storage of $O(\kappa n_a + n_a n)$, and a good-case round complexity of 10 rounds. With Byzantine dealers, the round complexity reduces by 2 rounds due to the removal of CS-Deliver. In terms of computation, all parties perform $O(n^2)$ operations. While this protocol incurs higher costs compared to the original protocol, it provides increased fault tolerance.

We present detailed security analysis in Section C.

7. Evaluation

We evaluate the performance of our protocol and compare it with the state-of-the-art standalone DKG protocol by Das et al. [29] for solving the DKG for use over blockchain, using their docker-based open-source implementation [27].

Implementation details. We implemented our protocol in Rust 1.84, comprising approximately 8,422 lines of code. To instantiate the verifiable encryption of committed messages, we employed the non-interactive class-group VSS [58].

TABLE 2: Committee sizes considered for our experiments.

Committee	Number of nodes				
Tribe Size (n)	64	96	128	192	256
Clan Size (n_c)	42	64	80	112	132
Family Size (n_a)	14	16	17	18	18

Additionally, we used Merkle trees as cryptographic accumulators. Our implementation is flexible and supports any reconstruction threshold $\ell \geq f$.

Our implementation integrates several libraries, including the BICYCL library [17] for class groups, the Miracl library for cryptographic operations such as elliptic curve cryptography and BLS signatures, the ed25519_dalek library [26] for Ed25519 signatures, and the tokio library [78] for network communication.

The secret and secret shares in our implementation are 256 bits in size, providing a security level of 128 bits. Each element in the class group is compressed to occupy 219 bytes [23]. For elliptic curve operations, we utilize the BLS12-381 curve. Each commitment is represented as a curve element, which requires 48 bytes when encoded.

We utilize the Moonshot [35] SMR protocol to implement the blockchain. Moonshot achieves a commit latency of 3 network hops and supports block proposals at each network hop. We used ED25519 signatures for authentication inside the consensus protocol.

Experimental setup. We carried out our evaluations on the Google Cloud Platform (GCP), distributing nodes evenly across eight distinct GCP regions: Tokyo, Montréal, Oregon, Singapore, Northern Virginia, Los Angeles, London, and Iowa. Each node was hosted on an e2-standard-2 instance, equipped with 2 vCPUs, 8 GB of memory, and up to 4 Gbps of network bandwidth. All nodes operated on Ubuntu 20.04. The blockchain is deployed across the same geographic regions, consisting of a committee of 96 nodes, each running on a separate machine.

We evaluate our implementation by considering various committee sizes ranging from 64 to 256 nodes. Table 2 shows the size of the clan and family committees considered in our experiments for each tribe size (n). The clan and family sizes are selected to ensure a failure probability of less than 10^{-9}. For each n we consider a reconstruction threshold of $\ell = 2f$. All reported timings are averages over 10 runs.

Optimization. In Step 3 of our protocol, the dealer waits for only $2f+1$ valid signatures before creating encryptions for the missing signatures. However, in practice, since the majority of nodes are honest, the dealer can wait for a timeout duration to collect all the signatures. Once the dealer has received all the signatures, it will not need to create encryptions or corresponding proofs for the missing signatures. As a result, the dealer only needs to forward the signatures for all n nodes, and the clan nodes will only need to verify the signatures without the need to verify any encryptions, leading to improved performance. We refer to this execution as the *optimistic* execution. Additionally, we consider the regular execution where the dealer waits

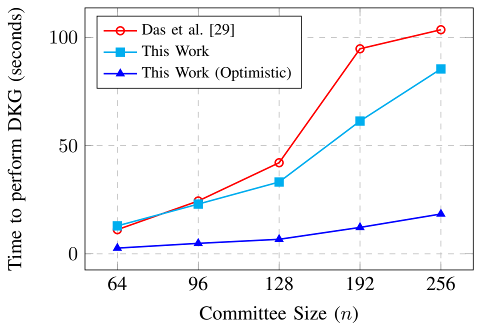

Figure 6: Comparison of time taken to perform DKG for use over blockchain.

for $2f+1$ signatures in Step 3 and sends encryptions for the missing signatures. Both of these cases are evaluated separately in our experiments.

Remark on Das et al. [29] DKG. The DKG protocol by Das et al. [29] is a standalone asynchronous DKG protocol. It requires a common source of randomness (aka common coin) to ensure termination in the asynchronous setting [44]. However, their protocol avoids relying on common coins when Byzantine parties do not actively attack the underlying consensus protocol, allowing it to terminate in a constant number of rounds. This execution is termed *good-case coin-free* execution. In such scenarios, their protocol incurs 18 rounds to terminate. To ensure the final DKG public key appears on the blockchain, an additional 4 rounds are needed when using the Moonshot consensus protocol, resulting in a total of 22 rounds in the good-case. Our evaluation considers only the good-case coin-free execution for their protocol.

7.1. Evaluation Results

Runtime. For our protocol, we measure the time from the protocol's initiation to the point where $2f+1$ parties successfully output their secret keys and individual threshold public keys. In contrast, for Das et al. [29], we measure the time from the protocol's initiation to when a supermajority of parties in the blockchain outputs the public key. The observed average times are illustrated in Figure 6. In the optimistic scenario, no encryptions are generated, leading to significantly improved performance. Our protocol outperforms the ADKG protocol by Das et al. [29] across all system sizes even when encryptions are generated and verified for f parties. This improvement is primarily attributed to our protocol requiring fewer communication rounds. Although the difference in round complexity between our protocol and theirs is significant, the expensive nature of class group encryptions for creation and verification results in a performance improvement that is slightly less than anticipated.

Runtime with parallel processing. To further assess the scalability of our protocol, we conducted additional experiments on GCP e2-highcpu-32 instances, each equipped

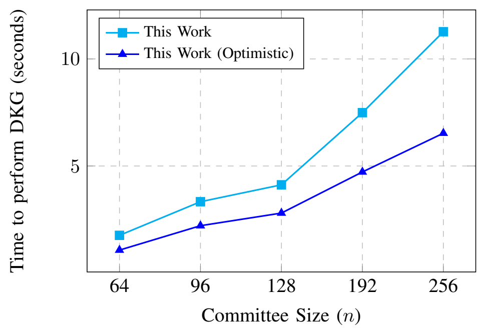

Figure 7: Time taken to perform DKG with parallel processing.

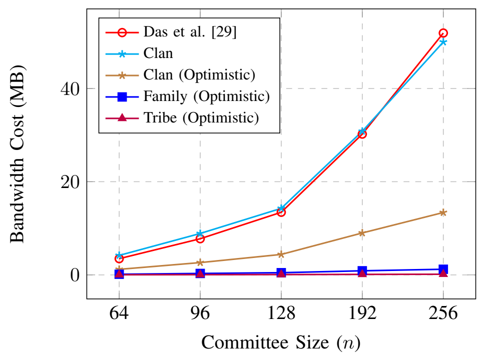

Figure 8: Comparison of bandwidth cost per node.

with 32 vCPU cores, 32 GB of RAM, and 16 Gbps network bandwidth. These specifications are representative of hardware commonly deployed in blockchain networks [9], [77]. By enabling parallel processing, our protocol achieved substantial performance improvements, as illustrated in Figure 7. For instance, with a committee size of 256 nodes, our protocol takes 6.53 seconds in the optimistic scenario and 11.27 seconds in the normal scenario, achieving a speedup of approximately 2.8 times and 7.6 times, respectively, compared to single-threaded execution. These gains stem from the parallelization of computationally intensive tasks, such as generating and verifying class group encryptions and verifying BLS signatures, allowing the protocol to efficiently handle larger system sizes. This enhancement underscores our protocol's ability to leverage modern multi-core architectures, making it highly suitable for large-scale blockchain deployments.

Bandwidth usage. We also measure the bandwidth usage of both protocols. For our protocol, we evaluate two scenarios: the optimistic scenario and the scenario where the dealers must compute encryptions for f parties. In both cases, the parties outside the clan send minimal data, resulting in significantly lower bandwidth usage. In the optimistic scenario, since the dealer clan does not need to send encryptions or their corresponding proofs, the bandwidth consumption is reduced. However, when encryptions and proofs are required, the bandwidth consumption for clan nodes becomes comparable to the bandwidth overhead per party in Das et al.'s [29] DKG protocol.

Scheme	Committee size (n)				
	64	**96**	**128**	**192**	**256**
Kate et al. [58]	871.41	1922.68	3387.67	7544.01	13349.01
Das et al. [29]	5.14	7.71	10.28	15.41	20.55
This work	2.59	2.98	3.61	4.24	4.77

TABLE 3: Comparison of Ethereum gas costs (in $\times 10^6$ gas units) for different committee sizes. Kate et al. [58] require each node to post the complete encrypted dealing and proof on-chain, Das et al. [29] require the nodes to post the final public key on-chain, and our protocol requires only family nodes to post an aggregate public key, an accumulation hash and a BLS multi-signature on-chain.

Blockchain storage. We compute the gas costs for existing standalone DKG protocols based on the amount of data they must post on-chain to complete the DKG process and have their public key appear on-chain when using the Ethereum blockchain. For this evaluation, we implemented a minimal Solidity contract (compiler version 0.8.26) to post randomized data matching the on-chain requirements of each protocol. Gas costs were measured using the Remix IDE [72] with the Remix VM (Cancun) environment, calculating the total gas required to store this representative data for various committee sizes. In Table 3, we provide associated gas costs for Kate et al. [58], Das et al. [29], and our protocol. It is noteworthy that the gas costs for Cascudo et al. [22] are identical to those of Kate et al. [58]. Similarly, other ADKG protocols would incur gas costs similar to Das et al. [29], as they follow similar mechanisms to publish their public key on-chain.

8. Conclusion

We designed an efficient discrete-log-based DKG protocol for blockchain, leveraging the blockchain's native consensus mechanism and random beacons. Our asynchronous protocol operates independently of network timing assumptions and is adaptable to various blockchains. Additionally, it allows us to solve DKG while tolerating up to one-half Byzantine faults within the DKG committee. Under the super-honest majority assumption, our protocol achieves sub-cubic communication, sub-quadratic computation, and minimal on-chain storage, and can terminate in just 11 rounds during good-case executions.

We experimentally evaluated our protocol in a geo-distributed setting with up to 256 nodes, observing faster termination and significantly reduced bandwidth overhead for most parties in the system. Furthermore, due to minimal on-chain storage, our protocol incurs significantly lower gas costs, as compared to the state of the art, when deployed on the Ethereum blockchain, highlighting its advantages. Finally, we also built and evaluated a multi-threaded implementation for our DKG system and demonstrated that DKGs can scale very well for hundreds of nodes and can be employed in the large-scale blockchain applications.

References

[1] Ittai Abraham, Gilad Ashsarov, Arpita Patra, and Gilad Stern. Asynchronous agreement on a core set in constant expected time and more efficient asynchronous vss and mpc. In *Theory of Cryptography Conference*, pages 451–482. Springer, 2025.

[2] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.

[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Annual International Cryptology Conference*, pages 39–70. Springer, 2023.

[4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 363–373, 2021.

[5] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

[6] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 331–341, 2021.

[7] Nicolas Alhaddad, Mayank Varia, and Ziling Yang. Haven++: Batched and packed dual-threshold asynchronous complete secret sharing with applications. *Cryptology ePrint Archive*, 2024.

[8] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. High-threshold avss with optimal communication complexity. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 479–498. Springer, 2021.

[9] Aptos. Node requirements. https://aptos.dev/en/network/nodes/validator-node/node-requirements, 2025. Accessed: 2025-03-12.

[10] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. Compressed σ-protocols for bilinear group arithmetic circuits and application to logarithmic transparent threshold signatures. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 526–556. Springer, 2021.

[11] Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. Grandline: adaptively secure dkg and randomness beacon with (log-) quadratic communication complexity. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 941–955, 2024.

[12] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. pages 429–459, 2021.

[13] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3502–3524, 2021.

[14] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, pages 353–380. Springer, 2020.

[15] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2002.

[16] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of cryptology*, 21(2):149–177, 2008.

[17] Cyril Bouvier, Guilhem Castagnos, Laurent Imbert, and Fabien Laguillaumie. I want to ride my bicycl: Bicycl implements cryptography in class groups. Cryptology ePrint Archive, Paper 2022/1466, 2022. https://eprint.iacr.org/2022/1466.

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

[19] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pages 123–132, 2000.

[20] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 98–116. Springer, 1999.

[21] Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*, pages 537–556. Springer, 2017.

[22] Ignacio Cascudo and Bernardo David. Publicly verifiable secret sharing over class groups and applications to dkg and yoso. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 216–248. Springer, 2024.

[23] Guilhem Castagnos, Dario Catalano, Fabien Laguillaumie, Federico Savasta, and Ida Tucker. Two-party ecdsa from hash proof systems and efficient instantiations. In *Annual International Cryptology Conference*, pages 191–221. Springer, 2019.

[24] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[25] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. In *34th International Symposium on Distributed Computing*, 2020.

[26] Dalek. Dalek cryptography. http://dalek.rs, 2025. Accessed: 2025-01-09.

[27] Sourav Das. Prototype implementation for the paper practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. https://github.com/sourav1547/htadkg.

[28] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. pages 2502–2517, 2022.

[29] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *USENIX Security Symposium*, 2023.

[30] Sourav Das, Zhuolun Xiang, and Ling Ren. Powers of tau in asynchrony. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024.

[31] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. Verifiable secret sharing simplified. *Cryptology ePrint Archive*, 2023. Accepted in IEEE SP'25.

[32] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *S&P'22*, pages 2518–2534. IEEE, 2022.

[33] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. GearBox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. pages 683–696, 2022.

[34] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989.

[35] Isaac Doidge, Raghavendra Ramesh, Nibesh Shrestha, and Joshua Tobkin. Moonshot: Optimizing chain-based rotating leader bft via optimistic proposals. In *International Conference on Dependable Systems and Networks (DSN)*, 2024.

[36] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[37] Drand. Drand - a distributed randomness beacon daemon. https://github.com/drand/drand.

[38] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[39] Paolo D'Arco and Douglas R Stinson. On unconditionally secure robust distributed key distribution centers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 346–363. Springer, 2002.

[40] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[41] Hanwen Feng, Zhenliang Lu, and Qiang Tang. Dragon: Decentralization at the cost of representation after arbitrary grouping and its applications to sub-cubic dkg and interactive consistency. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing*, pages 469–479, 2024.

[42] Hanwen Feng, Tiancheng Mai, and Qiang Tang. Scalable and adaptively secure any-trust distributed key generation and all-hands checkpointing. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 2636–2650. ACM, 2024.

[43] Hanwen Feng and Qiang Tang. Asymptotically optimal adaptive asynchronous common coin and dkg with silent setup. *Cryptology ePrint Archive*, 2024.

[44] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[45] Matthias Fitzi and Martin Hirt. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 163–168, 2006.

[46] Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In *Public Key Cryptography: 4th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2001 Cheju Island, Korea, February 13–15, 2001 Proceedings 4*, pages 300–316. Springer, 2001.

[47] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20:51–83, 2007.

[48] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*, pages 51–68, 2017.

[49] Jacob Gorman, Lucjan Hanzlik, Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Pratik Sarkar, and Sri AravindaKrishnan Thyagarajan. Vraas: Verifiable randomness as a service on blockchains. *Cryptology ePrint Archive*, 2024.

[50] Jens Groth. Non-interactive distributed key generation and key resharing. *Cryptology ePrint Archive*, 2021.

[51] Christoph U Günther, Sourav Das, and Lefteris Kokoris-Kogias. Practical asynchronous proactive secret sharing and key refresh. *Cryptology ePrint Archive*, 2022.

[52] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 147–176. Springer, 2021.

[53] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[54] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 322–340. Springer, 2005.

[55] Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. *Cryptology ePrint Archive*, 2004.

[56] Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 119–128. IEEE, 2009.

[57] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. *IACR Cryptol. ePrint Arch.*, 2012:377, 2012. https://eprint.iacr.org/2012/377.

[58] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. Non-interactive VSS using class groups and application to DKG. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, pages 4286–4300. ACM, 2024.

[59] Jonathan Katz. Round-optimal, fully secure distributed key generation. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VII*, volume 14926 of *Lecture Notes in Computer Science*, pages 285–316. Springer, 2024.

[60] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1751–1767, 2020.

[61] Torus Lab. Torus: Globally accessible public key infrastructure for everyone. https://tor.us/, 2021.

[62] Aptos Labs. Roll with move: Secure, instant randomness on aptos. Accessed on May 27,2025.

[63] Aptos Labs. Roll with move: Secure, instant randomness on aptos, 2024. Accessed on December 24, 2024.

[64] Mysten Labs. Unlocking the power of native randomness on sui. Accessed on May 27,2025.

[65] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[66] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and communication networks*, 9(17):4585–4595, 2016.

[67] Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, pages 275–292, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[68] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[69] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Advances in Cryptology—EUROCRYPT'91: Workshop on the Theory and Application of Cryptographic Techniques Brighton, UK, April 8–11, 1991 Proceedings 10*, pages 522–526. Springer, 1991.

[70] randao.org. Blockchain based verifiable random number generator, 2017. Accessed on Jan 22, 2024.

[71] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[72] Remix. Remix project jump into web3. https://remix-project.org/, 2025. Accessed: 2025-01-22.

[73] Supra Research. On-chain randomness fulfillment via verifiable random functions. Accessed on May 27,2025.

[74] Victor Shoup. Sing a song of simplex. In *38th International Symposium on Distributed Computing*, 2024.

[75] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[76] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. *IACR Communications in Cryptology*, 1(2), 2024.

[77] Sui. Sui validator node configuration. https://docs.sui.io/guides/operator/validator-config, 2025. Accessed: 2025-03-12.

[78] Tokio. Tokio: An async runtime for rust. https://tokio.rs/, 2025. Accessed: 2025-01-09.

[79] Dennis Trautwein, Aravindh Raman, Gareth Tyson, Ignacio Castro, Will Scott, Moritz Schubotz, Bela Gipp, and Yiannis Psaras. Design and evaluation of ipfs: a storage layer for the decentralized web. In *Proceedings of the ACM SIGCOMM 2022 Conference*, pages 739–752, 2022.

[80] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[81] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. hbacss: How to robustly share many secrets. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022*. The Internet Society, 2022.

[82] Haibin Zhang, Sisi Duan, Chao Liu, Boxin Zhao, Xuanji Meng, Shengli Liu, Yong Yu, Fangguo Zhang, and Liehuang Zhu. Practical asynchronous distributed key generation: Improved efficiency, weaker assumption, and standard model. In *2023 53rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 568–581. IEEE, 2023.

Appendix A. Extended Preliminaries

A.1. Linear erasure and error correcting codes.

- Encode. Given inputs $m_1, \ldots, m_b$, an encoding function Encode computes $(u_1, \ldots, u_n) = \mathsf{Encode}(m_1, \ldots, m_b)$, where $(u_1, \ldots, u_n)$ are code words of length n. A combination of any b elements of the code word uniquely determines the input message and the remaining of the code word.
- Decode. The function Decode computes $(m_1, \ldots, m_b) = \mathsf{Decode}(u_1, ..., u_n)$, and is capable of tolerating up to c errors and d erasures in code words $(s_1, \ldots, s_n)$, if and only if $n - b \geq 2c + d$.

A.2. Cryptographic accumulators.

Formally, given a parameter κ, and a set $\mathbf{D}$ of n values $d_1, \ldots, d_n$, an accumulator has the following components:

- $\mathsf{Gen}(1^\kappa, n)$: This algorithm takes as input a security parameter κ (represented in unary form 1^κ) and an accumulation threshold n (the maximum number of values that can be securely accumulated), and outputs an accumulator key a_κ. The accumulator key a_κ is part of the q-SDH setup and is publicly accessible to all parties.
- $\mathsf{Eval}(a_\kappa, \mathbf{D})$: This algorithm takes an accumulator key a_κ and a set $\mathbf{D}$ of values to be accumulated, and outputs an accumulation value z for the set $\mathbf{D}$.
- $\mathsf{CreateWit}(a_\kappa, z, d_i, \mathbf{D})$: This algorithm takes an accumulator key a_κ, an accumulation value z for $\mathbf{D}$, and a value d_i; it returns $\perp$ if $d_i \notin \mathbf{D}$ and a witness w_i if $d_i \in \mathbf{D}$.
- $\mathsf{Verify}(a_\kappa, z, w_i, d_i)$: This algorithm takes an accumulator key a_k, an accumulation value z for $\mathcal{D}$, a witness w_i and a value d_i, returns true if w_i is the witness for $d_i \in \mathcal{D}$, and false otherwise.

  This algorithm takes an accumulator key a_κ, an accumulation value z for $\mathbf{D}$, a witness w_i, and a value d_i; it returns true if w_i is the correct witness for $d_i \in \mathbf{D}$ and false otherwise.

In this work, we use bilinear accumulator from Nyugen [67] which satisfies the following property:

Lemma 1 (Collision-free accumulator [67]). *The bilinear accumulator is collision-free. That is, for any set of size n and a probabilistic polynomial-time adversary $\mathcal{A}$, the following function is negligible in κ:*

$$\mathsf{P}r\left[\begin{array}{c|c} ak \leftarrow \mathsf{Gen}(1^\kappa, n), & \\ (\{d_1, \ldots, d_n\}, d', w') \leftarrow & (d' \notin \{d_1, ..., d_n\}) \wedge \\ \mathcal{A}(1^\kappa, n, ak), & (\mathsf{Verify}(ak, z, w', d') = 1) \\ z \leftarrow \mathsf{Eval}(ak, \{d_1, \ldots, d_n\}) & \end{array}\right]$$

A.3. SCRAPE test [21]

In this work, we employ Feldman commitments to commit to the evaluations of the secret-sharing polynomial. To verify the degree of the commitment efficiently, we utilize the low-degree test introduced in SCRAPE [21]. The protocol employs the $\mathsf{PC.DegCheck(cmt)}$ interface, which samples a random degree $n - f - 2$ polynomial $z(.)$ in $\mathbb{F}$ and outputs 1 if the following condition holds:

$$\prod_{i \in [n]} \mathsf{cmt}[i]^{z(i) \cdot \lambda_i} = 1_{\mathbb{G}} \tag{5}$$

where $\lambda_i = \prod_{j \in [n], j \neq i} \frac{1}{(i-j)}$; otherwise outputs 0.

A.4. Blockchain Functionality

We present the simplified ledger/blockchain from Gearbox [33] in Fig. 9, which provides a simple, easy-to-use timed ledger functionality for submitting messages on a global ledger or a blockchain. The functionality $\mathcal{F}_{\mathsf{BC}}$ and the parties have access to a global ticker functionality [12] to keep track of time. The adversary $\mathcal{A}$ can corrupt an arbitrary number of participating parties P_i and run on their behalf. Additionally, $\mathcal{A}$ has the power to order the submitted messages for appending on the ledger. The blockchain functionality guarantees the following security properties:

1) **Persistence.** Suppose parties P_i and P_j are honest, and $\mathsf{TO}_i^{\mathsf{sid}}$ and $\mathsf{TO}_j^{\mathsf{sid}}$ are outputs of (Get, sid) obtained by P_i

and P_j respectively at different times. Then either $\mathsf{TO}_i^{\mathsf{sid}}$ is a prefix of $\mathsf{TO}_j^{\mathsf{sid}}$ or vice versa.

2) **Bounded timestamps** When a message is submitted to $\mathcal{F}_{\mathsf{BC}}$ at time t (obtained via the ticker functionality [12]), the functionality guarantees that it eventually gets appended to the ledger by the adversary by time $t' \leq t + \Delta$. Additionally, it appears on every honest party P_j's ledger $\mathsf{TO}_j^{\mathsf{sid}}$ by time $t' + \Delta$ on issuing a (Get, sid) command by P_j. This property ensures that an adversary can delay appending the message by at most Δ time where Δ is a bounded delay known to the adversary and the functionality. It also guarantees the liveness of the ledger as all submitted transactions will eventually get appended.

Initialization:
For party P_i set $\mathsf{TO}_i = ()$ and $\mathsf{Sent}^{\mathsf{sid}} = \emptyset$.

Interface for party P_i:
Input: $(Send, \mathsf{sid}, m)$

1) Output (P_i, sid, m) to the adversary.
2) Add (P_i, m, t^{now}) to $\mathsf{Sent}^{\mathsf{sid}}$.

Input: (Get, sid)

1) Output (Get, P_i) to the adversary.
2) Return $\mathsf{TO}_i^{\mathsf{sid}}$

Interface for Adversary:
Input: (Add, sid, m, t)

1) Append (m, t) to $\mathsf{TO}_i^{\mathsf{sid}}$.

At any time, $\mathcal{F}_{\mathsf{BC}}$ automatically enforces these restrictions:
Persistence: If P_i and P_j are honest, then either $\mathsf{TO}_i^{\mathsf{sid}}$ is a prefix of $\mathsf{TO}_j^{\mathsf{sid}}$ or $\mathsf{TO}_j^{\mathsf{sid}}$ is a prefix of $\mathsf{TO}_i^{\mathsf{sid}}$.

Liveness and bounded timestamps: For all messages $(\mathsf{sid}, m, t) \in \mathsf{Sent}^{\mathsf{sid}}$, there is a $t' \leq t + \Delta$ such that by time $t' + \Delta$, we have $(m, t') \in \mathsf{TO}_i^{\mathsf{sid}}$ for all honest P_i.

Figure 9: Simplified Blockchain functionality $\mathcal{F}_{\mathsf{BC}}$ (from Gearbox [33]).

A.5. Randomness Beacon

We recall the formal definition of a randomness beacon from [28] below.

Definition 2 (Randomness Beacon)**.** *A randomness beacon RB [28] is defined using a tuple of four algorithms:*

- $\mathsf{Gen}(1^\kappa) \rightarrow (\mathsf{vk}, \mathsf{sk})$ *: Takes as input the security parameter κ and outputs a key pair* $(\mathsf{vk}, \mathsf{sk})$*, where* vk *is the public verification key and* sk *is the secret key.*
- $\mathsf{Eval}(\mathsf{sk}, x) \rightarrow (y, \pi)$ *: Takes as input the secret key* sk *and input x and outputs a random string $y \in \{0,1\}^\kappa$ and a proof π attesting to correct computation of y.*
- $\mathsf{Verify}(\mathsf{vk}, x, y, \pi) \rightarrow 1/0$*: Takes as input the verification key* vk*, input x, output y, and proof π, and outputs a bit denoting whether the proof verifies or not.*

A tuple of algorithms denoted as $\mathit{RB} = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Verify})$ is a randomness beacon if it fulfills:

- ***Correctness**. For any $\kappa \in \mathbb{N}$, any* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$*, and any input x,* if $(y, \pi) \leftarrow \mathsf{Eval}(\mathsf{sk}, x)$ then $\mathsf{Verify}(\mathsf{vk}, x, y, \pi) \rightarrow 1$.
- ***Unpredicatability**. For any $\kappa \in \mathbb{N}$,* $(\mathsf{vk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^\kappa)$*, and any set of n valid input-output pairs $\{x_i, y_i, \pi_i\}_{i \in [n]}$, a PPT adversary $\mathcal{A}$ cannot predict the output on an unqueried input x'. More formally, for $\forall n \in \mathbb{N}$, the following holds:*

$$\Pr\left[y' == y'' \middle| (x', y', \pi') \leftarrow \mathcal{A}(\mathsf{vk}, \{x_i, y_i, \pi_i\}_{i \in [n]}) \right.$$
$$\left. \wedge (y'', \pi'') \leftarrow \mathsf{Eval}(\mathsf{sk}, x')\right] \leq negli(\kappa),$$

where (x', y'') has not been computed by RB.

Appendix B. Additional Related Work

Synchronous DKG protocols. Synchronous DKG protocols have been a topic of research for decades [11], [20], [41], [47], [66], [69], [76]. A majority of these protocols assume the availability of a broadcast channel, which ensures consensus within a predefined time frame. However, practical blockchains operate in a non-synchronous environment, where such strict timing guarantees are not feasible. Consequently, existing blockchains cannot be used to realize the broadcast channel required by these protocols.

Pedersen [69] proposed the first DKG protocol, utilizing a non-interactive information-theoretic verifiable secret sharing scheme. However, Gennaro et al. [47] demonstrated that Pedersen's protocol allows an attacker to bias the public key distribution, leading them to propose a scheme that eliminates this vulnerability, albeit at a higher computational cost. Neji et al. [66] introduced a simpler mechanism to mitigate the bias attack. Canetti et al. [20] extended Gennaro et al.'s work [47], making it secure against adaptive adversaries. Gurkan et al. [52] designed a PVSS-based DKG protocol that produces a public verification transcript of linear size. However, a significant limitation of their protocol is that the secret key is a group element rather than a field element, rendering their approach incompatible with standard threshold signature or encryption schemes widely used in practice.

Recently, several works [11], [41], [76] have proposed DKG protocols designed for point-to-point network settings without assuming broadcast channels. Shrestha et al. [76] achieve $O(\kappa n^3)$ communication complexity with expected $O(1)$ rounds while Feng et al. [41] and Bacho et al. [11] attain sub-cubic communication complexity. However, this comes at the cost of linear round complexity. Furthermore, both Feng et al. [41] and Bacho et al. [11] rely on stronger cryptographic primitives, such as zk-SNARKs, and assume non-standard assumptions like the one-more discrete log assumption (OMDL) to ensure adaptive security [11].

Concurrent works. Concurrent to our work, Abraham et al. [1] proposed a nearly quadratic communication ADKG protocol with adaptive security, optimal resilience, and expected $O(\kappa)$ round complexity. However, their protocol is not designed to generate keys for discrete-log-based cryptosystems. It also requires stronger setup assumptions, such as the use of verifiable random functions (VRFs), and relies on stronger cryptographic assumptions, including OMDL assumption and AGM.

Similarly, Feng et al. [43] introduced an adaptively secure ADKG protocol with quadratic communication complexity and expected $O(1)$ round complexity. However, their protocol depends on a stronger setup assumption, specifically a silent setup phase, which incurs cubic communication overhead.

Appendix C. Security Analysis of DKG over blockchain

Claim 1. *If an honest party P_i invokes* $\mathsf{Deliver}$ *with a value m corresponding to the accumulation value z, then, except with negligible probability of error, all honest parties will eventually receive the value m.*

Proof. Suppose an honest party P_i invokes $\mathsf{Deliver}$ for a value m associated with the accumulation value z. To do so, P_i must have sent a valid tuple $\langle \mathsf{codeword}, s_j, w_j \rangle$, derived from the value m, to the j^{th} party in $\mathcal{C}$.

Upon receiving the tuple $\langle \mathsf{codeword}, s_j, w_j \rangle$, the j^{th} party forwards its code word to all other parties in $\mathcal{C}$. Consequently, all parties in $\mathcal{C}$ will receive at least $f_c + 1$ valid code words, which are sufficient to decode the value m. However, these guarantees rely on $\mathcal{C}$ having an honest majority, a condition that fails with only negligible probability of error. □

Claim 2 (Communication Complexity of $\mathsf{Deliver}$)**.** *Let κ denote the size of the accumulator and w represent the size of the witness. Invoking* $\mathsf{Deliver}$ *with an input of size ℓ incurs a communication complexity of $O(n_c\ell + wn_c^2)$.*

Proof. When a $\mathsf{Deliver}$ operation is invoked on an object of size ℓ, each party in the $\mathcal{C}$ multicasts a code word of size $O(\ell/n_c)$, along with a witness of size w. Consequently, the total communication complexity amounts to $O(n_c\ell + wn_c^2)$. □

Claim 3. *When a consortium of at least $f_c + 1$ senders in the $\mathcal{C}$ invoke* $\mathsf{CS\text{-}Deliver}$ *to propagate the common value m, then all honest parties in $\mathcal{P}$ will eventually receive the value m, except with negligible probability of error.*

Proof. Suppose a consortium of at least $f_c + 1$ senders P_i invokes $\mathsf{CS\text{-}Deliver}$ to propagate a value m. Each of these $f_c + 1$ senders must have transmitted a valid tuple $\langle \mathsf{codeword}, s_j, w_j, z \rangle$, derived from the value m, to the party $P_j \in \mathcal{P}$.

As a result, each honest party $P_j \in \mathcal{P}$ must have received a consistent and valid code word s_j from at least f_c+1 parties. Party P_j then forwards the tuple $\langle \mathsf{codeword}, s_j, w_j, z \rangle$ to all honest parties. Consequently, each honest party P_i will eventually receive at least $2f + 1$ valid code words corresponding to the accumulation value z, which are sufficient to decode the value m. □

Claim 4 (Communication Complexity of $\mathsf{CS\text{-}Deliver}$)**.** *Let κ denote the size of the accumulator and w represent the size of the witness. When a consortium of senders invokes* $\mathsf{CS\text{-}Deliver}$ *on an input of size ℓ, the communication complexity is $O(n\ell + (\kappa + w)n^2)$.*

Proof. Each party P_i that invokes $\mathsf{CS\text{-}Deliver}$ sends a code word of size $O(\ell/n)$, along with a witness of size w and an accumulator of size κ, to all parties in $\mathcal{P}$. This results in a communication cost of $O(\ell + (\kappa + w)n)$. For n_c parties, the total communication is $O(n_c\ell + (\kappa + w)n_c n)$. In the second step, all parties in $\mathcal{P}$ send a code word of size $O(\ell/n)$, along with a witness of size w and an accumulator of size κ, to all other parties. This results in a communication complexity of $O(n\ell+(\kappa+w)n^2)$. Thus, the total communication complexity is $O(n\ell + (\kappa + w)n^2)$. □

Claim 5. *If an honest party sends an* Ack *for a meta-DKG instance* MDKG_k*, then for each commitment* $\mathsf{cmt}_j \in \mathsf{MDKG}_k$*, (i) all honest parties have either received valid secret shares or their valid secret shares are encrypted in $\vec{E}_{I_j}$, and (ii) at least one honest party in $\mathcal{C}$ has received the pair $(\mathsf{cmt}_j, \vec{E}_{I_j})$, except with negligible probability of error.*

Proof. An honest party P_i sends an Ack for a meta-DKG instance MDKG_k only upon receiving an accompanying LQC, which contains at least f_c+1 signatures for each commitment $\mathsf{cmt}_j \in \mathsf{MDKG}_k$. The presence of $f_c + 1$ signatures for a commitment cmt_j implies that at least one honest party has received σ_j and verified that it includes $2f + 1$ valid signatures on cmt_j and verified encrypted secret shares $\vec{E}_{I_j}$ for any missing valid signatures. Consequently, this ensures that all honest parties have received either valid secret shares or that their encrypted secret shares are included in $\vec{E}_{I_j}$ for each $\mathsf{cmt}_j \in \mathsf{MDKG}_k$. Furthermore, it guarantees that at least one honest party in $\mathcal{C}$ has received the pair $(\mathsf{cmt}_j, \vec{E}_{I_j})$. These guarantees, however, do not hold if $\mathcal{C}$ has a dishonest majority, which occurs with only negligible probability of error. □

Lemma 2. *Let $(g^x, z_{mk}, \mathsf{AC}(z_{mk}))$ be the output from the SMR. All honest parties will receive aggregated commitment* cmt *and final secret key, except with a negligible probability of error.*

Proof. Note that $\mathsf{AC}(z_{mk})$ consists of an Ack message from at least one honest party in $\mathcal{C}$, with high probability. By Claim 5, all honest parties must have either received the secret shares corresponding to each commitment $\mathsf{cmt}_j \in \mathsf{MDKG}_k$, or their encrypted secret shares are included in $\vec{E}_{I_j}$. Additionally, at least one honest party in $\mathcal{C}$ has received the pair $(\mathsf{cmt}_j, \vec{E}_{I_j})$. After the SMR outputs $(g^x, z_{mk}, \mathsf{AC}(z_{mk}))$, the honest parties in $\mathcal{C}$ invoke the $\mathsf{Deliver}$ function to propagate $(\mathsf{cmt}_j, \vec{E}_{I_j})$ for every $\mathsf{cmt}_j \in \mathsf{MDKG}_k$. Consequently, by Claim 1, all honest parties in $\mathcal{C}$ will eventually receive $(\mathsf{cmt}_j, \vec{E}_{I_j})$ for each $\mathsf{cmt}_j \in \mathsf{MDKG}_k$.

Consequently, each honest party in $\mathcal{C}$ aggregates the commitments and missing encryption vectors to compute the aggregated commitment cmt and the aggregated encryption vector for missing shares, $\vec{E}$. Let $\mathcal{Q}$ denote the list of dealers included in MDKG_k. Each party P_i in $\mathcal{C}$ then combines the secret shares received from dealers in $\mathcal{Q}$ with the decryption of $\vec{E}[i]$ to derive its final secret key x_i.

Additionally, the parties in $\mathcal{C}$ invoke the CS-Deliver function to disseminate $(\mathsf{cmt}, \mathcal{Q})$ and distribute the corresponding $\vec{E}[i]$ to each party $P_i \in \mathcal{P}$. By Claim 3, each party $P_i \in \mathcal{P}$ will receive $(\mathsf{cmt}, \mathcal{Q})$ and at least $f_c + 1$ consistent shares of $\vec{E}[i]$. Subsequently, each party P_i combines the secret shares obtained from dealers in $\mathcal{Q}$ with the decrypted values of $\vec{E}[i]$ to compute its final secret key x_i. These guarantees hold unless $\mathcal{C}$ contains a dishonest majority, an event that occurs with only negligible probability. □

Lemma 3. *Let x denote the final secret key and x_i the final individual secret key for party P_i. All honest parties agree on the same public key g^x and the corresponding threshold public keys g^{x_i} for all $i \in [n]$, except with a negligible probability of error.*

Proof. By the agreement property of the underlying SMR or blockchain, all parties output a common value $(g^x, z_{mk}, \mathsf{AC}(z_{mk}))$, ensuring that all honest parties agree on the same public key g^x. Furthermore, by Lemma 2, all honest parties receive the same aggregated commitment cmt. Since these commitments correspond to the evaluations of the secret-sharing polynomial, the aggregated commitment cmt serves as the commitment to the individual secret key of each party. Consequently, the threshold public key for each party can be derived directly from cmt. □

Theorem 1 (Security). *Under the discrete-log hardness assumption, the random oracle model, and the assumption of verifiable encryption, the protocol Π_{DKG} in Figure 5 securely realizes $\mathcal{F}_{\mathsf{BDKG}}$ tolerating up to $f < n/3$ static Byzantine corruptions.*

Proof. Let $\mathcal{C} \subset [n]$ denote the set of parties corrupted by the adversary $\mathcal{A}$, where $|\mathcal{C}| \leq f$. We construct a simulator $\mathcal{S}$ that interacts with $\mathcal{F}_{\mathsf{BDKG}}$ and $\mathcal{A}$, ensuring that the view of $\mathcal{A}$ during its interaction with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{BDKG}}$ is indistinguishable from its view during the real protocol execution.

$\mathcal{S}$ acts as follows. It generates public and secret keys for all honest parties. Upon receiving (Gen, *sid*, P_j) from $\mathcal{F}_{\mathsf{BDKG}}$, where P_j is identified as an honest party, $\mathcal{S}$ performs the following steps. For all honest parties except the last one, it samples a random polynomial $p_j(.)$ of degree ℓ. It then distributes the individual secret shares $p_j(i)$ to each party P_i, for all $i \in [n]$, and sends the tuple $(\mathsf{cmt}_j, I_j, \sigma_j, \vec{E}_j, \pi_{I_j})$. Finally, it adds P_j to the set $\mathcal{Q}$.

When $\mathcal{A}$ posts the tuple $(\mathsf{cmt}_j, I_j, \sigma_j, \vec{E}_j, \pi_{I_j})$ on behalf of party $P_j \in \mathcal{C}$ before $\mathcal{S}$ has posted the message for the last honest party, $\mathcal{S}$ verifies the validity of the tuple. If the verification succeeds, $\mathcal{S}$ adds P_j to the set $\mathcal{Q}$. Subsequently, $\mathcal{S}$ defines $\widehat{y_i} = \prod_{j \in \mathcal{Q}} \mathsf{cmt}_j[i]$ for all $i \in [n]$. Using the values $p_j(i)$ previously sent to the honest party P_i, $\mathcal{S}$ reconstructs the polynomial $p_j(.)$ and further defines $\widehat{x_i} = \sum_{j \in \mathcal{Q}} p_j(i)$.

Now, $\mathcal{S}$ waits until $\mathcal{F}_{\mathsf{BDKG}}$ sends $(\mathsf{KEYS}, sid, \{y_j\}_{j \in [n]}, y, \{x_j\}_{j \in \mathcal{C}})$. It then computes the tuple for the last honest party P_j as follows: it defines $\mathsf{cmt}_j[i] = y_i \cdot \widehat{y_i}^{-1}\ \forall i \in [n]$ and samples a polynomial $p_j(.)$ of degree ℓ such that $p_j(i) = x_i - \widehat{x_i}$ for $P_i \in \mathcal{C}$. Using the zero-knowledge simulator, $\mathcal{S}$ computes the necessary proof π_{I_j} along with verifiable encryptions and posts the tuple as honest party P_j would.

Now for any corrupt party P_j that have still not posted anything, whenever $\mathcal{A}$ posts a message on behalf of them, $\mathcal{S}$ runs the verification of the message and if it passes, it uses $p_j(i)$ sent for all honest P_i to compute $p_j(.)$. Let $\mathcal{B}$ denote all corrupt parties that posted tuples that pass the verification after $\mathcal{S}$ published the tuple of the last honest party. Then $\mathcal{S}$ defines $p'(.) = \sum_{j \in \mathcal{B}} p_j(.)$ and sends $(\mathsf{BIAS}, sid, p'(.))$ to $\mathcal{F}_{\mathsf{BDKG}}$.

We demonstrate that the execution with $\mathcal{S}$ and $\mathcal{F}_{\mathsf{BDKG}}$ is indistinguishable from the execution of the real protocol Π_{DKG}. First, observe that all messages generated by $\mathcal{S}$ on behalf of honest parties are identical to those produced in the actual protocol, except for the final tuple from the last honest party. This final tuple ensures that the public keys and secret keys corresponding to the corrupted parties align precisely with those provided by the functionality. Moreover, due to the zero-knowledge property of the proof π_{I_j}, the proof is distributed indistinguishably from an honest proof.

At this stage, the adversary can deduce from the published information the current y, y_i for all parties, and x_i for the corrupted parties P_i, as provided by the functionality to $\mathcal{S}$. The adversary may then post additional secret sharing tuples for parties that have not yet participated. The simulator translates these tuples into a polynomial p' that represents the sum of the sharing polynomials for all corrupted parties that send their tuples after $\mathcal{S}$ has published the tuple for the last honest party.

By the simulation soundness of the proof and the security of the verifiable encryption, the adversary cannot use the y_i, encrypted shares, or NIZK proofs of sharing correctness from simulated honest parties to construct a y_i, encrypted shares, and valid proof of sharing correctness for a corrupted party that are correlated with a simulated honest party's y_i and shares. Consequently, the functionality updates the public and private keys exactly as the adversary would with these final messages, ensuring that the output matches that of the real protocol.

Finally, throughout the simulation, the simulator only aborts when a proof of a false statement provided by the adversary is accepted. However, the security guarantees of the protocol ensure that this event occurs with negligible probability. □