Anchor-DKG: Distributed Key Generation with Repeating Parties

Hanwen Feng*[1], Qiang Tang[2], and Sri AravindaKrishnan Thyagarajan[2]

[1]University of Canterbury, Christchurch, New Zealand, hanwen.feng@canterbury.ac.nz
[2]University of Sydney, Sydney, Australia, qiang.tang@sydney.edu.au,
aravind.thyagarajan@sydney.edu.au

Abstract

A party may participate in multiple threshold cryptosystems. For example, it may serve on multiple overlapping threshold committees in a proof-of-stake blockchain or a distributed oracle network, or act as a client of multiple cryptocurrency wallet services built on threshold cryptography. With conventional distributed key generation (DKG), each threshold system independently generates its key shares, imposing significant key-management overhead on such a *repeating party*. In contrast, modern key-management practice favors deriving all cryptographic material deterministically from a single master key, raising a fundamental question: *Can DKG be reconciled with key derivation while preserving security and compatibility with legacy threshold systems?*

We present *Anchor-DKG*, a new DKG protocol that allows up to t^{rec} (the reconstruction threshold) parties to deterministically fix their secret key shares while retaining standard security guarantees. Anchor-DKG supports concurrent executions with overlapping participants across multiple DKG instances and remains fully compatible with legacy threshold schemes, including ECDSA, BLS, Schnorr, and ElGamal.

At the core of Anchor DKG lies a new technique: *fixed-point distributed polynomial sampling (FpDpS)*. FpDpS allows parties to jointly sample a random $(t^{\mathsf{rec}}-1)$-degree polynomial f such that $f(i) = s_i$ at designated points i, where each s_i can be a private input, e.g., a key derived from a master secret. The final secret key remains $f(0)$, ensuring compatibility with existing discrete-log-based threshold systems. We provide an efficient construction of Anchor DKG under standard cryptographic assumptions, which, compared to classical constructions such as Gennaro et al. (J.Cryptol. 2007), only incurs one more point-to-point round and marginal computation. Experimental results show that, for a network size of $n = 128$, our protocol incurs a per-party computation cost of 1.59 s, compared to 1.36 s for GJKR.

1 Introduction

Threshold cryptography enables distributed trust, removing reliance on a single centralized entity. Over the years, threshold techniques have been developed for encryption [1], signatures [11, 33], randomness generation [23], and more, enabling a wide range of applications. In modern blockchain systems, threshold cryptography plays a central role in consensus protocols [2], cross-chain bridges [51],

*Part of this work was done while Hanwen was at the University of Sydney.

decentralized oracle networks [3], and in mitigating long-range attacks [29] and MEV attacks in Decentralized Finance [13].

A classical (n, t^{rec}) threshold cryptosystem begins with a setup phase in which a public key pk is generated and the corresponding secret key sk is shared among n participants so that any subset of t^{rec} shares can reconstruct sk. Without a trusted dealer to generate and distribute the shares, the n parties must run a distributed key generation (DKG) protocol to complete the setup. DKG is thus a cornerstone of threshold cryptography, and a substantial body of work (see, e.g., [8, 6, 28, 29, 38, 39, 27, 37]) has sought to improve it along various dimensions, including communication and round complexity, setup assumptions, and corruption thresholds.

Key management challenges in complex threshold systems. Prior work on distributed key generation has primarily focused on the efficiency and security of generating distributed keys for individual threshold cryptosystems. We investigate a complementary dimension: how to make these keys easier to *manage*, especially when a user participates in *multiple* threshold cryptosystems and therefore holds multiple secret key shares.

A natural example arises from cryptocurrency wallet services built on threshold cryptography. A user often needs to manage multiple cryptocurrency keys. In self-custody settings, such as those adopted by BitGo Wallet[1], the user may retain some of the shares themselves. This directly creates the burden of managing multiple independent secret values on the user side. The challenge is further amplified when some shares are intended to be kept on a "cold" device or even written down on paper for additional security, making the management of multiple key shares both cumbersome and error-prone.

Beyond cryptocurrency wallets, an increasing number of real-world systems deploy *overlapping* threshold cryptosystems. Examples include restaking protocols, such as EigenLayer [4], where parties may serve on committees for different decentralized applications, and decentralized oracle networks, such as Chainlink [3], where nodes may belong to multiple committees.

These scenarios reveal a common pattern: a party may repeatedly participate in multiple threshold cryptosystems and consequently hold multiple secret key shares. We refer to such a participant as a *repeating party*. Under conventional DKG, a repeating party involved in k threshold systems must execute k independent DKG instances and manage k distinct secret key shares, making secure key management a fundamental challenge in complex threshold systems.

Recommended practice in the non-threshold setting. Key storage and management is a long-standing challenge even in conventional, non-threshold settings. A standard approach is to protect critical secret keys using secure hardware modules, such as HSMs, TPMs, hardware wallets, or other secure elements. These mechanisms are widely used to provide confidentiality and availability for secret keys, and are promoted by major standards and regulatory frameworks such as FIPS 140-3 [45] and NIST SP 800-57 [26].

However, secure hardware is often constrained in the number of keys it can store persistently, due to the cost and limitations of tamper-protected memory. For instance, a TPM 2.0 chip may provide only one or two persistent key slots [7]. As a result, storing every application-level secret key directly inside secure hardware is often impractical, especially for users or services that must manage many cryptographic identities.

One possible workaround is to encrypt all secret keys under a master key and store the resulting ciphertexts outside the secure hardware. This, however, shifts part of the key-management problem

[1] `https://developers.bitgo.com/docs/wallet-types`

to the storage and availability of these ciphertexts. An attacker may render them unavailable, for example through ransomware attacks [52]. Mitigating such availability risks typically requires additional operational measures, such as maintaining redundant copies across different platforms.

A more common and scalable approach is to use key derivation functions (KDFs) [43, 53]. In this approach, the secure hardware stores a single *master* key, from which many application-specific keys are deterministically derived as needed. This provides compact key management while avoiding the need to store a large number of independent secret values. Such an approach is recommended by standards such as NIST SP 800-108 [24] and SP 800-56C [25].

Unfortunately, existing threshold cryptosystems are incompatible with this practice. In conventional DKG, a repeating party participating in k different threshold systems obtains k independent random shares. These shares are neither derived from a common master key nor compressible into one, forcing the party to manage multiple independent secret values even when the corresponding non-threshold setting would naturally rely on deterministic key derivation.

DKG-free solutions without "nativity". The key management challenge has been recognized in the literature, with recent proposals introducing DKG-free threshold signature [11, 33] and encryption schemes [34]. For example, the multiverse threshold signature scheme by Baird et al. [11] allows a user to generate a secret key locally and reuse it across multiple committees.

However, these approaches sacrifice compactness, producing signatures or ciphertexts larger than their single-party equivalents, and more critically, fail to meet the *nativity* requirement essential for real-world compatibility. In short, the output of a threshold scheme should be verifiable using exactly the same verification algorithms as its single-party counterpart, without protocol-specific modifications or custom verification logic. This property is vital in practice: for instance, threshold signature schemes that do not yield standard ECDSA or Schnorr signatures are unusable in Bitcoin, whose consensus layer enforces fixed verification logic. Likewise, Ethereum's BLS12-381 signature verification allows no deviation in structure or size. Even minor incompatibilities can block adoption or impose heavy engineering overhead. The NIST call for threshold cryptography [5] explicitly stresses the need for such native integration to enable drop-in adoption in legacy systems.

Can we formulate a variant of DKG and design an efficient protocol that addresses modern key management challenges while preserving full compatibility with legacy discrete-log-based schemes?

1.1 Our contribution

We answer this question affirmatively by defining and proposing a new protocol-Anchor-DKG, which brings together two foundational yet historically separate areas of cryptographic research: *distributed key generation* and *deterministic key derivation*. Our key insight is that, while each party must participate in the DKG protocol to ensure correctness and robustness, they can deterministically derive their own key share for each role from a single master key.

This integration enables nodes to scale across many committees or virtual roles without storing and securing a proliferation of unrelated keys. Anchor DKG preserves the standard discrete-logarithm structure ($\mathsf{sk} = x \in \mathbb{Z}_q$, $\mathsf{pk} = g^x$), ensuring compatibility with widely deployed cryptographic systems. As a result, it also satisfies the *nativity requirement*: the threshold output remains verifiable/compatible using unmodified single-party counterparts.

Conceptually, this work opens a new direction in threshold cryptography by embedding deterministic derivation to enable reusable key material, bringing structured key management to distributed settings without compromising security or compatibility.

We summarize our main contributions below.

- We introduce Anchor-DKG that enables compact key management in threshold cryptosystems. To rigorously capture robustness and full secrecy in concurrent settings, we develop security definitions that support streamlined concurrent security analysis. The Anchor-DKG protocol supports any network size n and reconstruction threshold $t^{\mathsf{rec}} < n$. An (n, t^{rec})-Anchor-DKG tolerates up to $n - t^{\mathsf{rec}}$ Byzantine corruptions and up to $t^{\mathsf{rec}} - 1$ total corruptions (either Byzantine or honest-but-curious, whose internal state is revealed to the adversary). Moreover, up to t^{rec} parties may be designated as *repeating parties*, which can deterministically derive their secret shares from a master key[2].
- Since distributed polynomial sampling is a core technique in conventional DKGs, we develop a novel method for realizing fixed-point distributed polynomial sampling (FpDpS), which allows the network to sample a polynomial with fixed evaluations at specified points while remaining random elsewhere. This technique we believe may be of independent interest.
- We present an efficient Anchor-DKG protocol that achieves concurrent full secrecy and robustness under the standard discrete logarithm assumption in the random oracle model, without relying on PKI or CRS setup. Compared to an optimized variant of GJKR [35]—a classical and widely recognized protocol offering the same security guarantees under identical assumptions[3]—our protocol introduces only marginal overhead. Communication-wise, both protocols have the same round complexity in the optimistic case, while ours incurs just one additional point-to-point round in the adversarial case; their broadcast costs are identical. Computation-wise, based on our Java implementation of both protocols, the per-node optimistic-case computation time is 1.59 s versus 1.36 s at $n = 128$, and 31 s versus 22.3 s at $n = 512$. For most practical deployments where n is at most a few tens, the overhead is negligible—for example, in DFINITY [22] with $n = 34$, the additional computation time is only 7 ms.

1.2 Broader Applications

Facilitating cryptocurrency-wallet backup. Anchor-DKG can simplify backup management in partially custodial threshold-wallet systems [4]. Consider a 2-out-of-3 wallet in which a user holds one primary share on a hot device, a service provider holds a co-signing share to enforce security policies such as spending limits or multi-factor authentication, and the user holds a second fallback share for recovery if the provider becomes unavailable. With Anchor-DKG, the user's shares can be deterministically derived from ordinary wallet seeds. The user can therefore back up these shares using familiar seed-backup practices, such as writing the seeds down on paper, without needing to update their personal backups after every DKG execution—a process that is easy to forget or perform incorrectly. Shares that cannot be derived in this way, such as the provider's non-repeating share, can still be refreshed after each DKG execution; this is acceptable because such shares are managed by a professional service provider.

Weighted threshold cryptosystems with compact key management. In weighted systems like Proof-of-Stake, a party's influence scales with its weight. A standard implementation strategy is *virtualization* [50, 29], where a party with weight w_i acts as w_i distinct virtual nodes. While this

[2] In practice, some parties naturally act as repeating parties while others do not. As discussed in the Appendix A.3, allowing parties to "register" as repeating may further incentivize sustained participation.

[3] Despite its age, as shown in Table 1, GJKR remains one of the most practical designs for DKG with full secrecy.

[4] We thank an anonymous CCS reviewer for suggesting this application.

avoids the heavy computation of non-linear secret sharing [32], it requires storing w_i independent key shares, leading to unmanageable storage costs for high-weight entities. Anchor-DKG resolves this via deterministic key derivation: a party can generate all w_i virtual shares on the fly from a single master secret, reducing local storage from linear to constant.

Multi-factor key derivation/management. The Multi-Factor Key Derivation Function [44, 18, 49] was recently proposed to enable more resilient key derivation and more flexible key management, by incorporating multiple popular authentication factors into the process. In those settings, popular factors could be *pre-determined* biometrics instead of freshly generated randomness, thus conventional DKG protocols are not applicable. Anchor-DKG may offer alternative path constructing and analyzing security of those applications: our core technique of fixed-point distributed polynomial sampling (FpDpS) naturally facilitates a key/randomness generation protocol from multiple secrets, some of which are pre-determined. Extra requirements in those applications also motivate studies on Anchor-DKG, with all repeating parties (i.e., allowing all shares to be predetermined), and being non-interactive. We leave those as interesting open problems.

2 Technique Overview

In this section, we will give a brief overview of how we address the challenges in security formalization and building an efficient construction for Anchor-DKG.

2.1 Our Modeling Choices

Let us first discuss the following key questions in formulating Anchor-DKG as a new primitive and overview our modeling choices, which aim to enable a diverse range of applications.

On the admissible parameters. An Anchor-DKG instance allows parties to derive their secret shares from a master key via the KDF, thus greatly reducing the key management overhead for those parties. Although it would be ideal for all parties to enjoy this privilege, it is not difficult to observe that the number of these privileged parties (referred to as *repeated parties*, in line with the primary application of Anchor-DKG in the overlapping setting) must be bounded by the reconstruction threshold t^{rec}. Otherwise, it would be infeasible for more than t^{rec} externally specified values to lie on the same polynomial of degree $(t^{\mathsf{rec}} - 1)$.

A larger t^{rec} is thus needed to accommodate more repeating parties. Consequently, it has effects on the number of corrupted parties the protocol can tolerate. For robustness (which ensures that honest parties can obtain expected outputs), the protocol can tolerate at most $n - t^{\mathsf{rec}}$ Byzantine corrupted parties. On the other hand, a high reconstruction threshold t^{rec} enhances secrecy (ensuring that the adversary cannot learn the secret key) even if the internal states of $t^{\mathsf{rec}} - 1$ parties are leaked. To capture this benefit, we explicitly model both Byzantine and honest-but-curious corruption: the total number of corrupted parties must be less than t^{rec}, while the number of Byzantine corruptions is bounded by $n - t^{\mathsf{rec}}$.

We remark that, even without supporting repeating parties, many applications (e.g., threshold wallets) may prioritize secrecy over robustness. Similarly, practical threshold signature protocols such as FROST [40] do not provide robustness. Our modeling therefore offers a flexible framework accommodating diverse requirements.

On the necessity of specifying key derivation. The core feature of Anchor-DKG is that repeating parties are allowed to exert some control over their secret shares, rather than simply receiving random shares from the protocol. Our definition must therefore explicitly capture how repeating parties determine their shares. A seemingly natural approach is to allow a repeating party to input a value, such that an Anchor-DKG protocol outputs this value as its secret share. This would leave the key derivation function (KDF) as an external component, offering modularity.

However, to ensure meaningful secrecy, all secret shares of honest parties (repeating or not) must be pseudorandom and computationally independent. Allowing arbitrary inputs is risky: for instance, if two honest repeating parties input the same value as their shares, and $t^{\mathsf{rec}} - 1$ other shares leak to the adversary, the adversary could reconstruct the entire secret polynomial!

To enforce pseudorandomness and independence in the inputs of honest repeating parties and to prevent misuse, we incorporate the KDF as an *integral part* of Anchor-DKG. In this formulation, repeating parties input only their master keys, and thus the derived shares guarantee the desired properties.

On concurrent security definitions. Applications of Anchor-DKG naturally involve scenarios where multiple DKG instances—across potentially overlapping committees—are executed concurrently. Moreover, repeating parties reuse the same master key across different instances, which may lead to severe security issues if concurrent security is not carefully addressed.

A conventional approach to ensuring concurrent security is through the UC framework [14], which, however, may be unnecessarily heavy. Furthermore, the presence of cross-instance secrets (i.e., the master key) requires even more intricate modeling, such as the joint-state UC framework [16].

Instead, we adopt a simpler and more user-friendly oracle-based approach from [30]. In this framework, the adversary's ability to create and participate in other protocol instances is captured by an interactive oracle. Informally, our concurrent security definition ensures that, even when the adversary can interact with this oracle, it cannot break the security of the challenge instance, as long as the number of corrupted parties in that instance does not exceed the threshold. Furthermore, this guarantee continues to hold even if the adversary concurrently invokes arbitrary threshold cryptosystems (e.g., signature or encryption schemes) that use secret keys derived from other instances, since the interactive oracle already grants the adversary access to the secret keys of those instances.

To formalize the oracles, we follow [41] in defining an Anchor-DKG protocol as a sequence of round functions. A key advantage of this oracle-based approach is that it enables a simpler concurrent security proof, as demonstrated in [30].

On the security goals. As Anchor-DKG is supposed to allow an honest repeating party to output its derived key as the secret share, we need to extend the *robustness* property to capture this requirement. Regarding secrecy, we choose to follow the strongest definition, *full secrecy* [35], which captures that the adversary can neither learn anything about the secret key beyond the public key nor bias the secret key, thus sufficing for nearly all Dlog based threshold cryptosystems [5]. The details are available in §3.

[5]Like *all* existing DKG protocols with full secrecy (without assuming erasures), we assume static adversary.

2.2 Towards Our Solution

In the following, we overview our solutions. We first recall the conventional DKG design and walk through two natural attempts for realizing Anchor-DKG, each of which falls short either in security or efficiency. These limitations motivate our approach. We then discuss the subtleties involved in the security proofs and explain our treatments, ultimately leading to our final protocol.

Overview of existing DKG protocols. Most DKG protocols [47], [35], [15], [31], [39] follow a common paradigm where each party verifiably shares a secret to the network, ensuring every other honest party has a corresponding share. Specifically, in a (t^{rec}, n)-DKG, every party P_i uniformly samples a $(t^{\mathsf{rec}}-1)$-degree polynomial f_i, and verifiably delivers $f_i(j)$ to each P_j (via a verifiable secret sharing (VSS) protocol), such that when an honest P_j outputs $f_i(j)$, it has been convinced that all other honest parties output the corresponding evaluations of the same $(t^{\mathsf{rec}}-1)$-degree polynomial. Finally, each P_i obtains a secret key share $sk_i = \sum_{j\in\mathsf{Qual}} f_j(i)$, where Qual is the set of all parties who have correctly delivered shares to honest parties. Moreover, all secret key shares are consistent, as they are on the same $(t^{\mathsf{rec}}-1)$-degree polynomial $F(x) = \sum_{j\in\mathsf{Qual}} f_j(x)$.

Notably, as F is an aggregation of randomly sampled polynomials, its evaluations *cannot* be controlled by any party, prohibiting a deterministic share derivation.

Therefore, to overcome this challenge, we need a *fixed-point* distributed polynomial sampling (FpDpS) technique, such that the resulting F has fixed evaluations, e.g. $F(i) = s_i$, where each s_i is a pseudorandom value specified by the "repeating" party P_i, while otherwise remaining random.

A naive attempt for FpDpS. Recall that $F = \sum_{j\in\mathsf{Qual}} f_j$ has random evaluations as honestly sampled f_j's have random evaluations at every point. Therefore, a seemingly straightforward idea is to coordinate the local sampling process of each f_j, such that $\sum_{j\in\mathsf{Qual}} f_j(i) = s_i$, where s_i is determined by a repeating party P_i.

Note that the set Qual is decided *after* the secret sharing phase where each f_j has been determined. Thus, we have to stick to a simple coordination strategy where $\sum_{j\in\mathsf{Qual}} f_j(i)$ is always s_i, regardless of the choice of Qual, which means those f_j for $j \neq i$ will not have any impact on the aggregation result. Namely, we may set $f_i(i) = s_i$, the secret specified by P_i, and $f_j(i) = 0$ for any $j \neq i$; while except the constraints, all f_j's are random $(t^{\mathsf{rec}}-1)$-degree polynomials as usual. This approach ensures that the secret share transmitted to an honest repeating party is identical to the one specified by that party, as an honest party is always included Qual. It is also easy to enforce a party P_j to share with polynomial f_j satisfying $f_j(i) = 0$ for some $i \in [n]$, for example, by using Feldman VSS where all $g^{f(i)}$'s are public.

However, as prudent readers may have noticed, the restriction on f_j leaks too much information that allows the adversary to entirely reconstruct f_j. Specifically, assume the total number of corrupted parties is $(t_{\mathsf{rec}}-1)$, and there are two honest repeating parties P_{i_1} and P_{i_2}. In this case, notice that for every f_i of an honest P_i, the adversary knows at least t_{rec} evaluations of f_i, i.e., $(t_{\mathsf{rec}}-1)$ leaked to the corrupted parties, plus $f_i(i_1) = 0$ and/or $f_i(i_2) = 0$, which allows the adversary to reconstruct the polynomial f_i.

Fixing security. To address the above issue, (1) we must avoid letting an honest party P_j share a polynomial f_j whose $f_j(i)$ for a repeating party P_i is known to the adversary. On the other hand, (2) we still need to ensure that the final secret-sharing polynomial F, which is determined by $\{f_j\}_{j\in\mathsf{Qual}}$, has fixed evaluations at j's for all "valid" repeating P_j' s, regardless of the choice of Qual.

According to (1), $f_j(i)$ for any repeating P_i cannot be zero (or any other fixed value); while

due to (2), for any Qual set, $\{f_j\}_{j\in\mathsf{Qual}}$ can define a polynomial F such that $F(i) = s_i$. Since Qual cannot be predicted a-priori, we must introduce a certain "error correction" mechanism in the approach. We considered the following approach.

- In the first phase, every repeating party P_i verifiably shares its input secret s_i with a $(t^{\mathsf{rec}}-1)$ polynomial $\widehat{f}_i$ such that $\widehat{f}_i(0) = s_i$, such that all honest parties receive the corresponding evaluations of the polynomial.
- In the second phase, *every* party P_j *randomly* samples a polynomial f_j such that $f_j(i) = \widehat{f}_i(j)$ for all repeating $P_i \in \mathcal{R}_{\mathtt{sid}}$, and then it verifiably distributes the evaluations of f_j to the network. [6]
- Finally, similar to the first attempt, the network can decide the Qual set of secret sharings of the second phase; However, to ensure $F(i) = s_i$ for all honest repeating P_i, the final polynomial F is an *interpolation* of all "qualified" secret sharing polynomials, i.e.,

$$F(x) = \sum_{j\in\mathsf{Qual}'} \lambda_j(0) f_j(x), \tag{1}$$

  where Qual' is a subset of Qual and consists of t^{rec} elements, and $\lambda_j(x) = \prod_{k\in\mathsf{Qual}',k\neq j} \frac{x-k}{j-k}$ is a Lagrange polynomial.

It is easy to see $F(i) = s_i = \widehat{f}_i(0)$, as $F(i) = \sum_{j\in\mathsf{Qual}'} \lambda_j(0) f_j(i) = \sum_{j\in\mathsf{Qual}'} \lambda_j(0)\widehat{f}_i(j)$. At the same time, the adversary does not know $f_j(i)$ (for an honest party P_j and an honest repeating party P_i) as this value is secretly delivered to P_j.

Unfortunately, this approach incurs significant latency compared to conventional DKG protocols. Let Δ_{VSS} denote the rounds of a verifiable secret sharing (VSS) protocol. A conventional DKG costs Δ_{VSS} rounds, while our approach roughly *doubles* the rounds. Note that a VSS protocol usually incurs a few rounds of broadcast (that depends on concrete instantiations); therefore the additional phase of VSS will significantly affect the actual performance.

2.3 FpDpS: Our Final Approach

Our goal is to securely realize FpDpS, without introducing significant overhead compared to conventional distributed polynomial sampling. We observe that, in our second attempt, defining the final polynomial F as $F(x) = \sum_{j\in\mathsf{Qual}'} \lambda_j(0) f_j(x)$ already ensures invariance for repeating parties. Namely, even if the parties in the second phase directly send the shares *without using VSS*, so that there is no global agreement on which dealers have successfully shared their secrets, any subset of t^{rec} correct shares can still be used to reconstruct the secret inputs of correct parties.

Based on this observation, our idea is to "extend" the set of repeating parties so that *all* parties can benefit from the invariance. Specifically, suppose in the first phase there are t^{rec} repeating parties (denoted by $\{P_i\}_{i\in\mathcal{R}_{\mathtt{sid}}}$) who have verifiably shared their secret inputs using polynomials $\{\widehat{f}_i\}_{i\in\mathcal{R}_{\mathtt{sid}}}$. Then, in the second phase, the secret-sharing polynomial f_j of each party P_j must satisfy $f_j(i) = \widehat{f}_i(j)$ for all $i \in \mathcal{R}_{\mathtt{sid}}$, meaning that f_j is fully determined by $\{\widehat{f}_i\}$. In particular, this implies:

$$f_j(x) = \sum_{i\in\mathcal{R}_{\mathtt{sid}}} \lambda_i(x)\widehat{f}_i(j),$$

[6] Similarly, with Feldman VSS, one can verify this by comparing $g^{f_j(i)}$ and $g^{\widehat{f}_i(j)}$, which both are public information.

where $\{\lambda_i(x)\}_{i\in\mathcal{R}_{\mathtt{sid}}}$ are the Lagrange basis polynomials.

In this case, the "non-repeating" parties also enjoy the invariance, as any subset of t^{rec} "correct" shares from the second phase can be used to reconstruct the same share sk_z (the final secret share of a non-repeating party P_z):

$$\sum_{j\in\mathsf{LocQ}} \widetilde{\lambda}_j(0) f_j(z) = \sum_{j\in\mathsf{LocQ}} \widetilde{\lambda}_j(0)(\sum_{i\in\mathcal{R}_{\mathtt{sid}}} \lambda_i(z)\widehat{f}_i(j)) = \sum_{i\in\mathcal{R}_{\mathtt{sid}}} \lambda_i(z)\widehat{f}_i(0),$$

which is independent of LocQ, the set of "correct" shares received by each P_z. Here, $\{\widetilde{\lambda}_j\}_{i\in\mathcal{R}_{\mathtt{sid}}}$ are the Lagrange basis polynomials, w.r.t. the set LocQ.

We remark that non-repeating parties can efficiently check whether a share in the second phase is correct. In particular, assuming the repeating parties employ Feldman VSS to share their secret inputs, all parties can see the commitments $\{(g^{\widehat{f}_i(j)})_{j\in[0,n]}\}_{i\in\mathcal{R}_{\mathtt{sid}}}$. Then, a non-repeating party P_z can verify whether $s' = f_j(z)$ by checking $g^{s'} = \prod_{i\in\mathcal{R}_{\mathtt{sid}}}(g^{\widehat{f}_i(j)})^{\lambda_i(z)}$.

The remaining issue is how to ensure the existence of t^{rec} "qualified" repeating parties which have successfully shared their input secrets. Note that this *cannot* be achieved by enforcing t^{rec} repeating participants, as malicious repeating party may simply be silent, resulting in fewer "qualified" repeating parties, causing a liveness issue. Our solution is to have all parties as dealers in the first phase, such that at least t^{rec} qualified dealers can be guaranteed. Then, the network can apply any deterministic rule to select exactly t^{rec} qualified dealers (denoted by $\widetilde{R}$) such that all valid repeating parties are included, while other valid VSS instances will be ignored in the subsequent processes.

Our FpDpS approach can also be understood from the following angle. In the first phase, all repeating parties share their inputs s_i's to the network, while non-repeating parties share random values s_i's to the network. After the first phase, the final polynomial F has been determined such that $F(i) = s_i$ for all $i \in \widetilde{R}$, $|\widetilde{R}| = t^{\mathsf{rec}}$. The second phase is to help each P_j not in $\widetilde{R}$ obtain their shares as $F(j)$. Since all parties in $\widetilde{R}$ already know their shares, they do not need to receive any message in the second phase.

Moreover, among the t^{rec} points that define the polynomial F, at least one of them is from an honest party, which ensures the final F is unpredictable to the adversary, as the honest contribution is either fully random (when the party is non-repeating) or pseudorandom (when the party is repeating).

2.4 Challenges in Simulation and Our Treatments

While we can intuitively argue that the final polynomial F is unpredictable, there are barriers in the security proof that indicate subtle security issues which we must address carefully.

Simulation challenges. We first explain the essence of the challenge with the simple Feldman VSS. As we wish to demonstrate the adversary cannot learn anything about $F(0)$ (which is the final secret key) from the protocol execution, we need to construct a simulator Sim which does not have any information about $F(0)$ but can generate a simulated execution view for the adversary. Sim needs to provide messages to the adversary on the behalf of honest parties, while at least for one honest party (denoted as P_{i^*}) such that $i^* \in \widetilde{R}$, Sim must not know the polynomial $\widehat{f}_{i^*}$(otherwise, it knows F). A typical Sim performs the VSS on the behalf of P_{i^*} by broadcasting $(g^{\widehat{f}_{i^*}(j)})_{j\in[0,n]}$ and providing $\{\widehat{f}_*(j)\}_{j\in\mathcal{C}}$ to corrupted parties.

Then, in the second phase, Sim needs to provide evaluations of f_j to corrupted non-repeating parties P_z's, on the behalf of an honest P_j, which satisfies $g^{f_j(z)} = \prod_{i \in \mathcal{R}_{\mathtt{sid}}} (g^{\widehat{f}_i(j)})^{\lambda_i(z)}$. However, as Sim does not know $\widehat{f}_{i^*}(j)$, it has no way to compute $f_j(z)$ either, which causes a simulation failure!

Augmenting Pedersen commitment with simulatable opening. The root reason is that the polynomial $\widehat{f}_i$ is information theoretically determined by the public commitment. A natural treatment is to employ Pedersen VSS [46], which has a hiding polynomial $\widehat{f}'_i$ to conceal the information of $\widehat{f}_i$. Specifically, the commitment to $\widehat{f}_i$ is in the form of $(g^{\widehat{f}_i(j)} h^{\widehat{f}'_i(j)})_{j \in [0,n]}$, while each receiver P_j will be provided a pair $(\widehat{f}_i(j), \widehat{f}'_i(j))$, where h is an independent generator of the group.

Nonetheless, using Pedersen VSS alone does not solve the simulation challenge. Specifically, the simulator Sim may have simulated a commitment $(g^{\widehat{f}_{i^*}(0)} h^{\widehat{f}'_{i^*}(j)})_{j \in [0,n]}$ for an honest P_{i^*}, without knowing the openings for $\{g^{\widehat{f}_{i^*}(j)} h^{\widehat{f}'_{i^*}(j)}\}$ for honest P_j's. However, in the second phase, Sim still needs to provide openings to $g^{f_j(z)} h^{f'_j(z)} = \prod_{i \in \mathcal{R}_{\mathtt{sid}}} (g^{\widehat{f}_i(j)} h^{\widehat{f}'_i(j)})^{\lambda_i(z)}$, which it will not be able to do due to the lack of knowledge of $(\widehat{f}_{i^*}(j), \widehat{f}'_{i^*}(j))$.

To resolve this issue, we augment the Pedersen commitment to support simulatable opening. Specifically, in the second phase, an honest party P_j does not need to provide both $f_j(z)$ and $f'_j(z)$ to P_z. Instead, P_j will send $f_j(z)$ along with a non-interactive zero-knowledge proof of knowledge (NIZKPoK) π which demonstrates the knowledge of $f'_j(z)$. Note that $(f_j(z), \pi)$ can serve as an opening to $g^{f_j(z)} h^{f'_j(z)}$ due to the knowledge soundness of NIZKPoK, while the simulator Sim is able to arbitrarily open any group element, including $g^{f_j(z)} h^{f'_j(z)}$, by generating a simulated proof.

3 Security Model and Anchor-DKG Definition

We aim at developing distributed discrete-log cryptosystems with *compact secret* storage requirements. Specifically, we want to achieve the following features simultaneously.

- *Compact secret storage in repeating setting.* Each party should store only a constant-size secret, from which all necessary key shares across different instances can be deterministically derived.
- *Preservation of cryptographic structure.* The protocol must maintain compatibility with standard discrete-logarithm-based schemes, ensuring that threshold outputs remain verifiable using unmodified legacy algorithms.

Setup and Network Assumptions. Before presenting formal definitions of Anchor-DKG, we clarify the setup assumptions and notations used across this paper.

User participation and corruption. We consider a dynamic set of party identities, denoted by $\mathcal{U}$, and allow the adversary to introduce new users. Upon a user's introduction, the adversary determines its corruption status. We distinguish between two types of corruption:

- **Byzantine corruption.** A Byzantine party is controlled by the adversary and may arbitrarily deviate from the protocol specification. The set of Byzantine parties is denoted by $\mathcal{C}_{\mathtt{B}}$.
- **Honest-but-Curious (HbC) corruption.** An HbC party follows the protocol specification, but exposes all its internal states to the adversary. The set of HbC parties is denoted by $\mathcal{C}_{\mathtt{P}}$.

The set of honest parties $\mathcal{H}$ is always $\mathcal{U} \backslash (\mathcal{C}_{\mathtt{B}} \cup \mathcal{C}_{\mathtt{P}})$. This framework models dynamic user participation with static corruption. It offers greater flexibility than the standard static model—where the

participant universe and corruption sets are fixed at initialization—and better reflects real-world scenarios for threshold cryptosystems where users join over time.[7]

Instance setup. Each DKG instance is assigned a unique identifier $\mathtt{sid}$, and it is parameterized by the number of participants $n_{\mathtt{sid}}$ and the reconstruction threshold $t^{\mathsf{rec}}_{\mathtt{sid}}$. Its participants are a subset $\mathcal{P}_{\mathtt{sid}} \subset \mathcal{U}$, and each pair of honest parties in $\mathcal{P}_{\mathtt{sid}}$ can establish a secure channel, such that the adversary cannot learn the communication content between honest parties or tamper with it. In addition, all parties have access to a broadcast channel with a public delay bound Δ. The channel guarantees that if an honest party broadcasts a message at time T, then every other party receives it by time $T+\Delta$. It further guarantees consistency: all parties receive the same message even if the sender is malicious, and, when the sender is honest, this message is exactly the one that the sender broadcast. For the ease of description, we assume each party in $\mathcal{P}_{\mathtt{sid}}$ is assigned a per-instance ID in $[n_{\mathtt{sid}}]$.

Synchronization. All parties in $\mathcal{P}_{\mathtt{sid}}$ operate in a *synchronous* environment. Specifically, the parties have a synchronized clock, and there exists a public bound for message delivery between honest nodes. Thus, the DKG instance can proceed in rounds. Namely, all messages sent by an honest node can be delivered to an honest receiver node by the end of this round (i.e., the beginning of the next round), though the delivery time bound can vary across instances and communication channels.

Syntax. Aligning with most DKG works in the literature [39, 21], we focus on DKG formalisation for discrete-log-based threshold cryptosystems in which parties hold Shamir secret shares of the secret key. Namely, let $\mathbb{G}$ be a cyclic group with a prime order q, and let g be a generator of $\mathbb{G}$; There should be a $(t_{\mathsf{rec}}-1)$-degree polynomial $f \in \mathbb{Z}_p[X]$, such that $f(0)$ is the secret key $sk \in \mathbb{Z}_p$, and the public key is $pk = g^{sk} \in \mathbb{G}$, while each party P_i obtains a secret key share $sk_i = f(i)$. This setting already encompasses a wide range of legacy schemes, including threshold BLS, ECDSA, Schnorr, and ElGamal.

We can describe a $\mathsf{Anchor\text{-}DKG}$ scheme for the DLog key structure, $(pk = g^{sk} \in \mathbb{G}, sk \in \mathbb{Z}_q)$, with its main DKG protocol and two auxiliary functions for key derivation.

- MkGen. A user P_i runs the randomized algorithm to generate its master key msk_i.
- $\mathsf{ShareDeriv}(msk_i, \mathtt{sid})$. On inputs the master key msk_i of P_i and the identifier $\mathtt{sid}$ of a DKG instance, the deterministic algorithm outputs a secret key share $sk[\mathtt{sid}]_i$.
- $\mathsf{DKG}[\mathtt{sid}]$. A DKG instance is an interactive protocol parameterized by its participant set $\mathcal{P}_{\mathtt{sid}}$, consisting of repeating parties $\mathcal{R}_{\mathtt{sid}}$ and non-repeating parties $\mathcal{N}_{\mathtt{sid}}$, a reconstruction threshold $t^{\mathsf{rec}}_{\mathtt{sid}}$, and the timeout parameters $\vec{T}_{\mathtt{sid}}$. Each $P_i \in \mathcal{R}_{\mathtt{sid}}$ has its master key msk_i as input, while $P_i \in \mathcal{N}_{\mathtt{sid}}$ has no private input.

  At the end of the protocol, each $P_i \in \mathcal{P}_{\mathtt{sid}}$ outputs a threshold public key $pk[\mathtt{sid}]$, the set of public key shares $(pk[\mathtt{sid}]_j)_{j\in[n_{\mathtt{sid}}]}$, and a secret key share $sk[\mathtt{sid}]_i$. In a valid output, there exists a $(t^{\mathsf{rec}}_{\mathtt{sid}}-1)$-degree polynomial $f[\mathtt{sid}] \in \mathbb{Z}_q[X]$, such that

$$sk[\mathtt{sid}]_j = f[\mathtt{sid}](j),\ pk[\mathtt{sid}]_j = g^{sk[\mathtt{sid}]_j}$$

  for $j \in [n_{\mathtt{sid}}]$, and $pk[\mathtt{sid}] = g^{f[\mathtt{sid}](0)}$.

[7] Note that *all* existing DKG protocols with full secrecy (avoiding erasure assumptions) assume a static adversary, as overviewed in [39], so our model is not weaker than the state of the art. We discuss the challenges of realizing full adaptive security in Appendix A.2.

On the number of repeating parties. Since the secret key shares of repeating parties can be determined externally, the number of secret key shares belonging to repeating parties must not be greater than the reconstruction threshold. Otherwise, these shares cannot be on the same $(t^{\mathsf{rec}}_{\mathtt{sid}}-1)$-degree polynomial when there are more than $t^{\mathsf{rec}}_{\mathtt{sid}}$ repeating shares. Therefore, we have $|\mathcal{R}_{\mathtt{sid}}| \leq t^{\mathsf{rec}}_{\mathtt{sid}}$.

On the number of corrupted parties. On rough terms, a DKG protocol shall satisfy (1) secrecy, which means that the adversary cannot learn anything about the secret key beyond public information, and (2) robustness, which means that despite the presence of Byzantine parties, all honest nodes can obtain the expected outputs. To ensure the two properties, there are trivial upper bounds on the number of corrupted parties the protocol can tolerate.

- For secrecy, as any $t^{\mathsf{rec}}_{\mathtt{sid}}$ shares are sufficient to reconstruct the polynomial, the number of shares leaked to the adversary must be smaller than $t^{\mathsf{rec}}_{\mathtt{sid}}$. Namely, the total number of corrupted parties must be smaller than $t^{\mathsf{rec}}_{\mathtt{sid}}$, i.e.,

$$|(\mathcal{C}_{\mathtt{B}} \cup \mathcal{C}_{\mathtt{P}}) \cap \mathcal{P}_{\mathtt{sid}}| \leq t^{\mathsf{rec}}_{\mathtt{sid}} - 1, \tag{2}$$

  where $\mathcal{C}_{\mathtt{B}}$ is the set of Byzantine corrupted parties, and $\mathcal{C}_{\mathtt{P}}$ is the set of passively corrupted parties.

- For robustness, as the reconstruction of the secret requires $t^{\mathsf{rec}}_{\mathtt{sid}}$ shares, we will need at least $t^{\mathsf{rec}}_{\mathtt{sid}}$ parties to follow the protocol honestly. Namely,

$$n_{\mathtt{sid}} - |\mathcal{P}_{\mathtt{sid}} \cap \mathcal{C}_{\mathtt{B}}| \geq t^{\mathsf{rec}}_{\mathtt{sid}}. \tag{3}$$

Throughout this paper, we aim at designing protocols tolerating the maximal amount of corruptions. Specifically, when $t^{\mathsf{rec}} = t + 1$, where t is the number of Byzantine corrupted parties among all n parties, we only require $n \geq 2t + 1$ for both secrecy and robustness.

Remark 1. We emphasize that the above discussion does not mean that Anchor-DKG only supports a limited number of repeating parties. Instead, for any number of repeating parties, we can always set a larger reconstruction threshold t^{rec} to accommodate it. As we have also discussed, given a network of n parties, having larger t^{rec} means that the system can tolerate a larger number of all-type corruptions (honest-but-curious and Byzantine) for secrecy, while for robustness (or to successfully reconstruct), fewer Byzantine parties ($n - t^{\mathsf{rec}}$) can be tolerated. We remark that favoring secrecy over robustness is common in practice. For example, the FROST protocol [40], offers no robustness while ensuring secrecy for any $t^{\mathsf{rec}} < n$.

Concurrent Security. An Anchor-DKG protocol is *concurrently secure* if every instance DKG[sid], whose corruption is bounded as per Eq. 2 and Eq. 3, satisfies robustness and secrecy, even when the adversary concurrently creates and participates in other instances.

We model concurrent security following the framework of [30]. Our goal is not universal composability, which secures any protocol instance under arbitrary composition with other protocols, but rather to capture security when multiple instances of the *same* protocol run concurrently. In the oracle-based approach of [30], information the adversary learns from other instances is represented by oracles; a protocol is concurrently secure if a single instance remains secure even when the adversary has access to these oracles. Our definitions use oracles that explicitly allows the adversary to create new instances and obtain all information derivable from them. In particular, to capture the fact that an adversary may learn partial information about an instance's secret

key through subsequent executions of threshold cryptosystems that use this key, we conservatively grant the adversary access to all honest secret key shares of any non-challenging instance.

In the following, we first describe an Anchor-DKG protocol with a round function and an output function, which allows us to precisely define the messages the adversary can learn and how the adversary can influence the execution of an instance. Then, based on these notations, we introduce oracles to capture the adversary's ability of initialising and participating in other instances while simultaneously attacking an instance. Finally, we define the security properties for Anchor-DKG.

Round functions of Anchor-DKG. An Anchor-DKG protocol can be described with a round function and an output function. We express the round function as:

$$(\mathtt{st}_i, \mathtt{BM}, \overrightarrow{\mathtt{PMs}}) \text{ or } \perp \leftarrow \mathsf{Round}(\mathtt{sid}, i, \mathtt{RN}, \mathtt{st}_i, \overrightarrow{\mathtt{InBMs}}, \overrightarrow{\mathtt{InPMs}})$$

where $\mathtt{sid}$ is identifier of this instance, $i \in [n_{\mathtt{sid}}]$ is the identity of the party, $\mathtt{RN}$ is the number of the current round, $\mathtt{st}_i$ is the state of P_i, $\overrightarrow{\mathtt{InBMs}}$ is the sequence of all incoming broadcast messages from the last round, $\overrightarrow{\mathtt{InPMs}}$ is the sequence of all incoming point-to-point messages sent to P_i in the last round, $\mathtt{BM}$ is the broadcast message to be sent by P_i, and $\overrightarrow{\mathtt{PMs}}$ is the sequence of messages to be sent by P_i over point-to-point channel to different recipients. If the output is $\perp$, it indicates there are no enough valid messages to progress.

For $\mathtt{RN} = 1$, the input state $st_i = (msk_i, \mathtt{params})$ if $\mathtt{nid} \in \mathcal{R}$, or $st_i = (\mathtt{params})$ if $i \in \mathcal{N}_{\mathtt{sid}}$, where $\mathtt{params}$ are all public parameters of this instance. At the last round, each party invokes the following output function to finalize this instance.

$$(pk[\mathtt{sid}], (pk[\mathtt{sid}]_i)_{i\in[n_{\mathtt{sid}}]}, sk[\mathtt{sid}]_i) \text{ or } \perp \leftarrow \mathsf{Output}(\mathtt{sid}, i, \mathtt{st}_i, \overrightarrow{\mathtt{InBMs}}, \overrightarrow{\mathtt{InPMs}})$$

Here the output of $\perp$ indicates that P_i failed to generate an output.

Adding user oracle $\mathcal{O}_{\mathsf{AddU}}$. Recall that we allow the adversary to expand the participant universe $\mathcal{U}$ and specify the corruption status of new users upon their introduction. We formally capture this capability through the oracle $\mathcal{O}_{\mathsf{AddU}}$:

- **State maintenance:** The oracle maintains the participant universe $\mathcal{U}$, the identity sets for Byzantine parties ($\mathcal{C}_{\mathtt{B}}$), Honest-but-Curious parties ($\mathcal{C}_{\mathtt{P}}$), and honest parties ($\mathcal{H}$), as well as the set of master keys $\mathcal{MSK}$.
- **Query processing:** On input $(\mathtt{nid}, \mathtt{type})$ where $\mathtt{type} \in \{\mathtt{Byzantine}, \mathtt{HbC}, \mathtt{Honest}\}$: If $\mathtt{nid} \in \mathcal{U}$, ignore the query. Otherwise, update $\mathcal{U} \leftarrow \mathcal{U} \cup \{\mathtt{nid}\}$ and proceed as below:
  - If $\mathtt{type} = \mathtt{Byzantine}$: Update $\mathcal{C}_{\mathtt{B}} \leftarrow \mathcal{C}_{\mathtt{B}} \cup \{\mathtt{nid}\}$.
  - If $\mathtt{type} = \mathtt{HbC}$: Generate $msk_{\mathtt{nid}} \leftarrow \mathsf{MkGen}(1^\lambda)$, update $\mathcal{C}_{\mathtt{P}} \leftarrow \mathcal{C}_{\mathtt{P}} \cup \{\mathtt{nid}\}$ and $\mathcal{MSK} \leftarrow \mathcal{MSK} \cup \{(\mathtt{nid}, msk_{\mathtt{nid}})\}$, and **return** $msk_{\mathtt{nid}}$ to the adversary.
  - If $\mathtt{type} = \mathtt{Honest}$: Generate $msk_{\mathtt{nid}} \leftarrow \mathsf{MkGen}(1^\lambda)$, and update $\mathcal{H} \leftarrow \mathcal{H} \cup \{\mathtt{nid}\}$ and $\mathcal{MSK} \leftarrow \mathcal{MSK} \cup \{(\mathtt{nid}, msk_{\mathtt{nid}})\}$.

New instance oracle $\mathcal{O}_{\mathsf{newIns}}$. This is an interactive and stateful oracle that enables the adversary to create a new DKG instance, participate in the instance on the behalf of Byzantine corrupted parties, learn the states of honest-but-curious parties, and receive the messages of honest participants. We describe the oracle in Fig.1. At a high level, the adversary can query the oracle with an instance

initialization message $(\texttt{sid}, \texttt{init}, \mathcal{R}_{\texttt{sid}}, \mathcal{N}_{\texttt{sid}}, t^{\mathsf{rec}}_{\texttt{sid}})$, which specifies the unique identifier $\texttt{sid}$ of this instance, the participant information $\mathcal{R}_{\texttt{sid}}$ and $\mathcal{N}_{\texttt{sid}}$, and the reconstruction threshold $t^{\mathsf{rec}}_{\texttt{sid}}$. The oracle will create an instance with these parameters. Then, the adversary can send an instance execution message $(\texttt{sid}, \texttt{exe}, \{\texttt{BM}_{i'}, \overrightarrow{\texttt{PMs}}_{i'}\}_{i' \in \mathcal{C}_{\texttt{B}} \cap \mathcal{P}_{\texttt{sid}}})$ to the oracle, where $\{\texttt{BM}_{i'}, \overrightarrow{\texttt{PMs}}_{i'}\}_{i' \in \mathcal{C}_{\texttt{B}} \cap \mathcal{P}_{\texttt{sid}}}$ are messages that the adversary provides on the behalf of Byzantine corrupted nodes. The oracle then returns the messages generated on behalf of all honest or honest-but-curious parties for the current round, while the internal states of all honest-but-curious parties will be leaked to the adversary. Finally, if the created instance successfully generates a valid public key and secret key shares, and all repeating parties obtain secret key shares that are identical to the output of the share derivation function, the adversary receives the generated public key, public shares, and secret key shares of honest parties. Otherwise, the adversary gets the `Failed-Instance` message.

Remark 2. In our model, a newly created instance advances to the next round upon receiving an execution message $(\texttt{sid}, \texttt{exe}, \dots)$ from the adversary. This differs from the usual abstraction in which an instance progresses according to synchronous rounds. We adopt this modeling choice to capture realistic adversarial manipulation of the *pace* of different synchronous instances. In practice, timeout values may vary depending on network conditions, and an adversary may exploit this flexibility by causing one instance to progress much faster or slower than another.

An alternative modeling choice would be to let the adversary specify the timeout parameters when initiating an instance. This would prevent the adversary from adaptively controlling the pace of an instance after it starts, but it would also make the model unnecessarily complicated.

Challenge Oracle $\mathcal{O}_{\mathsf{Chall}}$. The adversary can access the challenge oracle *at most once* to create an instance that he intends to compromise. As described in Fig.1 (with boxed items), $\mathcal{O}_{\mathsf{Chall}}$ is identical to the new instance oracle $\mathcal{O}_{\mathsf{newIns}}$, except (1) that the corrupted parties in the participant set $\mathcal{P}_{\texttt{sid}} = \mathcal{R}_{\texttt{sid}} \cup \mathcal{N}_{\texttt{sid}}$ must be not greater than the corruption threshold, i.e., Equation 2 and 3 must be satisfied, and (2) that the adversary will not learn the secret shares of honest parties.

Concurrent Robustness. Even when an adversary participates in multiple DKG instances concurrently, in an instance with less-than-threshold corrupted parties, each party P_i can output the group public key $pk[\texttt{sid}]$, the public key shares $(pk[\texttt{sid}]_j)_{j \in [n_{\texttt{sid}}]}$, and its own secret key share $sk[\texttt{sid}]_i$. For honest parties, the outputs must satisfy: (1) Agreement on $pk[\texttt{sid}]$ and $(pk[\texttt{sid}]_j)_{j \in [n]}$; (2) Existence of a degree-$(t^{\mathsf{rec}} - 1)$ polynomial $F \in \mathbb{Z}_p[X]$ such that $sk[\texttt{sid}]_j = F(j)$, $pk[\texttt{sid}]_j = g^{sk[\texttt{sid}]_j}$, and $pk[\texttt{sid}] = g^{F(0)}$; (3) For $P_i \in \mathcal{R}_{\texttt{sid}}$ (a repeating party), $sk[\texttt{sid}]_i = \mathsf{ShareDeriv}(msk_i, \texttt{sid})$. The first two conditions are standard in DKG protocols; the third is unique to Anchor-DKG. Formally, we have the following definition.

Definition 1 (Concurrent robustness)**.** An Anchor-DKG scheme is concurrently robust, if for any security parameter $\lambda \in \mathbb{N}$, and any PPT adversary $\mathcal{A}$, it holds that

$$\Pr\begin{bmatrix} \mathcal{U}, \mathcal{H}, \mathcal{C}_{\texttt{B}}, \mathcal{C}_{\texttt{P}}, \mathcal{MSK}, \mathsf{InsIDs} \leftarrow \emptyset; \\ \mathcal{A}^{\mathcal{O}_{\mathsf{AddU}}, \mathcal{O}_{\mathsf{newIns}}, \mathcal{O}_{\mathsf{Chall}}}(1^\lambda) : \\ \mathcal{O}_{\mathsf{Chall}} \text{ returns } (\texttt{sid}, \texttt{final}, \texttt{Failed-Instance}) \end{bmatrix} \leq \mathsf{negl}(\lambda), \tag{4}$$

where $\mathcal{A}^{\mathcal{O}_{\mathsf{AddU}}, \mathcal{O}_{\mathsf{newIns}}, \mathcal{O}_{\mathsf{Chall}}}(1^\lambda)$ means that the adversary is allowed to query $\mathcal{O}_{\mathsf{AddU}}$, $\mathcal{O}_{\mathsf{newIns}}$ and $\mathcal{O}_{\mathsf{Chall}}$ in an arbitrary order, while we restrict the adversary to query $\mathcal{O}_{\mathsf{Chall}}$ with at most one instance initialization message. The winning condition of $\mathcal{A}$ is to make the challenge instance abort, i.e., $\mathcal{O}_{\mathsf{Chall}}$ returns $(\texttt{sid}, \texttt{final}, \texttt{Failed-Instance})$.

$\mathcal{O}_{\mathsf{newIns}}(\mathcal{MSK}, \mathsf{InsIDs}, \cdot)$ and $\boxed{\mathcal{O}_{\mathsf{Chall}}(\mathcal{MSK}, \mathsf{InsIDs}, \cdot)}$

// The program inside the oracle runs the following instructions once their conditions are met

Upon receiving $(\mathtt{sid}, \mathtt{init}, \mathcal{R}_{\mathtt{sid}}, \mathcal{N}_{\mathtt{sid}}, t^{\mathsf{rec}}_{\mathtt{sid}})$, **do**

  if $\mathtt{sid} \in \mathsf{InsIDs}$, **then** Ignore this message // Ensure every instance having a unique identifier

  $\mathcal{P}_{\mathtt{sid}} \leftarrow \mathcal{N}_{\mathtt{sid}} \cup \mathcal{R}_{\mathtt{sid}}, n_{\mathtt{sid}} \leftarrow |\mathcal{P}_{\mathtt{sid}}|$. $\boxed{\textbf{if Eq.2 or Eq.3 is not satisfied, then Ignore this message}}$

  $\mathtt{params}_{\mathtt{sid}} = (\mathtt{sid}, \mathtt{init}, \mathcal{R}_{\mathtt{sid}}, \mathcal{N}_{\mathtt{sid}}, t^{\mathsf{rec}}_{\mathtt{sid}}), \mathsf{InsIDs} \leftarrow \mathsf{InsIDs} \cup \{\mathtt{sid}\}$;

  $\mathtt{abort}_{\mathtt{sid}} \leftarrow \mathtt{false}$ // Use $\mathtt{abort}_{\mathtt{sid}}$ to track whether this instance aborts

  for $\mathsf{rn} \in [N], i, i_s, i_r \in \mathcal{P}_{\mathtt{sid}} = \mathcal{N}_{\mathtt{sid}} \cup \mathcal{R}_{\mathtt{sid}}$

    // Only a repeating party's initial state includes its master key msk_i

    if $i \in \mathcal{R}_{\mathtt{sid}} \cap (\mathcal{H} \cup \mathcal{C}_{\mathtt{P}})$ **then** $st_i \leftarrow (\mathtt{params}_{\mathtt{sid}}, msk_i)$ **else** $st_i \leftarrow \mathtt{params}_{\mathtt{sid}}$

    // Initialize all broadcast messages $\mathbf{BMs}$ and point-to-point messages $\mathbf{PMs}$ to be $\perp$

    Set $\mathbf{BMs}[\mathtt{sid}][\mathsf{rn}][i] \leftarrow \perp, \mathbf{PMs}[\mathtt{sid}][\mathsf{rn}][i_s][i_r] \leftarrow \perp$

  $\mathsf{RN}_{\mathtt{sid}} \leftarrow 0$ // Track the current round number

  return $(\mathtt{sid}, \mathtt{init\text{-}success})$

Upon receiving $(\mathtt{sid}, \mathtt{exe}, \{\mathtt{BM}_{i'}, \overrightarrow{\mathtt{PMs}}_{i'}\}_{i' \in \mathcal{C}_{\mathtt{B}} \cap \mathcal{P}_{\mathtt{sid}}})$ // Do the followings when $\mathcal{A}$ sent the execution message

  Update $\mathsf{RN}_{\mathtt{sid}} \leftarrow \mathsf{RN}_{\mathtt{sid}} + 1$

  if $\mathsf{RN}_{\mathtt{sid}} > N + 1$, **then** Ignore this message

  if $\mathsf{RN}_{\mathtt{sid}} > 1$ // Record the messages as those sent by Byzantine nodes in the last round

  then Update $\mathbf{BMs}[\mathtt{sid}][\mathtt{RN}_{\mathtt{sid}} - 1][i'] \leftarrow \mathtt{BM}_{i'}, (\mathbf{PMs}[\mathtt{sid}][\mathtt{RN}_{\mathtt{sid}} - 1][i'][i''])_{i'' \in (\mathcal{H} \cup \mathcal{C}_{\mathtt{P}}) \cap \mathcal{P}_{\mathtt{sid}}} \leftarrow \overrightarrow{\mathtt{PMs}}_{i'}, \forall i' \in \mathcal{C}_{\mathtt{B}} \cap \mathcal{P}_{\mathtt{sid}}$

  for $i \in (\mathcal{H} \cup \mathcal{C}_{\mathtt{P}}) \cap \mathcal{P}_{\mathtt{sid}}$ // Read the incoming messages for each honest node

    Set $\overrightarrow{\mathtt{InBMs}} \leftarrow (\mathbf{BMs}[\mathtt{sid}][\mathsf{RN}_{\mathtt{sid}} - 1][i'])_{i' \in \mathcal{P}_{\mathtt{sid}}}, \overrightarrow{\mathtt{InPMs}} \leftarrow (\mathbf{PMs}[\mathtt{sid}][\mathsf{RN}_{\mathtt{sid}} - 1][i'][i])_{i' \in \mathcal{P}_{\mathtt{sid}}}$

  if $\mathsf{RN}_{\mathtt{sid}} < N + 1$ // When this instance has not reached to the end of the final round.

    for $i \in (\mathcal{H} \cup \mathcal{C}_{\mathtt{P}}) \cap \mathcal{P}_{\mathtt{sid}}$ // Execute the round function

      Run $(st_i, \mathtt{BM}, \overrightarrow{\mathtt{PMs}}) \leftarrow \mathsf{Round}(\mathtt{sid}, i, \mathsf{RN}_{\mathtt{sid}}, st_i, \overrightarrow{\mathtt{InBMs}}, \overrightarrow{\mathtt{InPMs}})$ // Run Round for honest P_i

      if Round outputs $\perp$, **do** $\mathtt{abort}_{\mathtt{sid}} \leftarrow \mathtt{true}$

      Update $\mathbf{BMs}[\mathtt{sid}][\mathsf{RN}_{\mathtt{sid}}][i] \leftarrow \mathtt{BM}, (\mathbf{PMs}[\mathtt{sid}][\mathsf{RN}_{\mathtt{sid}}][i][i'])_{i' \in \mathcal{P}_{\mathtt{sid}}} \leftarrow \overrightarrow{\mathtt{PMs}}$ // Record messages sent by P_i

    // Send messages for all corrupted parties and the internal states of passively corrupted parties to the adversary

    return $(\mathtt{sid}, \mathsf{RN}_{\mathtt{sid}}, \{\mathbf{BMs}[\mathtt{sid}][\mathsf{RN}_{\mathtt{sid}}][i], \mathbf{PMs}[\mathtt{sid}][\mathsf{RN}_{\mathtt{sid}}][i][i_r]\}_{i \in (\mathcal{H} \cup \mathcal{C}_{\mathtt{P}}) \cap \mathcal{P}_{\mathtt{sid}}, i_r \in (\mathcal{C}_{\mathtt{B}} \cup \mathcal{C}_{\mathtt{P}}) \cap \mathcal{P}_{\mathtt{sid}}}, \{st_i\}_{i \in \mathcal{C}_{\mathtt{P}} \cap \mathcal{P}_{\mathtt{sid}}})$

  if $\mathsf{RN}_{\mathtt{sid}} = N + 1$

    for $i \in (\mathcal{H} \cup \mathcal{C}_{\mathtt{P}}) \cap \mathcal{P}_{\mathtt{sid}}$ // Execute the output function

      Run $(pk[\mathtt{sid}], (pk[\mathtt{sid}]_z)_{z \in [n]}, sk[\mathtt{sid}]_i) \leftarrow \mathsf{Output}(\mathtt{sid}, i, st_i, \overrightarrow{\mathtt{InBMs}}, \overrightarrow{\mathtt{InPMs}})$

    Interpolate a $(t^{\mathsf{rec}}_{\mathtt{sid}} - 1)$-degree polynomial $F[\mathtt{sid}]$ based on the first $t^{\mathsf{rec}}_{\mathtt{sid}}$ secret key shares generated above

    // Check the validity of outputs as below:

    if $\exists i_1, i_2 \in \mathcal{P}_{\mathtt{sid}} \cap \mathcal{H}, s.t., P_{i_1}$ and P_{i_2} output different public keys

      $\vee\ \exists i \in \mathcal{R}_{\mathtt{sid}} \cap \mathcal{H}, \mathsf{ShareDeriv}(msk_i, \mathtt{sid}) \neq sk[\mathtt{sid}]_i$ // Check if repeating parties can derive their shares

      $\vee\ \exists i \in \mathcal{P}_{\mathtt{sid}} \cap \mathcal{H}, s.t., \mathsf{Output}(\mathtt{sid}, i)$ returns $\perp$, or $pk[\mathtt{sid}] \neq g^{F[\mathtt{sid}](0)}$, // Check if all shares on $F[\mathtt{sid}]$

        or $pk[\mathtt{sid}]_j \neq g^{F[\mathtt{sid}](j)}$ for any $j \in [n_{\mathtt{sid}}]$, or $sk[\mathtt{sid}]_i \neq F[\mathtt{sid}](i)$

        then $\mathtt{abort}_{\mathtt{sid}} \leftarrow \mathtt{true}$

    if $\mathtt{abort}_{\mathtt{sid}} = \mathtt{true}$ **return** $(\mathtt{sid}, \mathtt{final}, \mathtt{Failed\text{-}Instance})$

      else return $(\mathtt{sid}, \mathtt{final}, \{pk[\mathtt{sid}], (pk[\mathtt{sid}]_j)_{j \in [n_{\mathtt{sid}}]}\}, \boxed{(sk[\mathtt{sid}]_j)_{j \in \mathcal{H} \cap [n_{\mathtt{sid}}]}})$

Figure 1: $\mathcal{O}_{\mathsf{newIns}}$ and $\mathcal{O}_{\mathsf{Chall}}$ for N-round Anchor-DKG. Items and items are only for $\mathcal{O}_{\mathsf{Chall}}$ and $\mathcal{O}_{\mathsf{newIns}}$, respectively

Concurrent full secrecy. Regarding secrecy, we extend the standard full secrecy [35] in the concurrent setting, which captures that an adversary can neither learn any information about the secret key beyond the public key nor bias the public key distribution. Specifically, this property is captured by a simulator Sim, which on input a uniformly sampled public key $y \leftarrow\!\!\$\ \mathbb{G}$ can simulate an execution of the protocol, such that the simulated execution is indistinguishable, in the view of the adversary, with a real execution that generates $pk = y$. In the concurrent setting, we additionally allow the adversary to participate in other instances concurrently.

To facilitate a formal definition, we view a simulator Sim for a Anchor-DKG scheme as an interactive algorithm with the following inputs and outputs.

- It takes as initial input a group element $y \in \mathbb{G}$, generators $g, h \in \mathbb{G}$, the instance information $(\mathtt{sid}, \mathtt{init}, \mathcal{R} * \mathsf{init}, \mathcal{N} * \mathsf{sid}, t^{\mathsf{rec}} * \mathtt{sid})$, the corruption information $(\mathcal{C} * \mathtt{B}, \mathcal{C} * \mathtt{P})$, and the master keys of all honest-but-curious parties $msk_i * i \in \mathcal{C}_{\mathtt{P}}$. It then returns $(\mathtt{sid}, \mathtt{init\text{-}success})$.
- Subsequently, for up to $N + 1$ times, it takes inputs of the form

$$(\mathtt{sid}, \mathtt{exe}, \{\mathtt{BM}_{i'}, \overrightarrow{\mathtt{PMs}}_{i'}\}_{i' \in \mathcal{C}_{\mathtt{B}} \cap \mathcal{P}_{\mathtt{sid}}}),$$

  where $\{\mathtt{BM}_{i'}, \overrightarrow{\mathtt{PMs}}_{i'}\}_{i' \in \mathcal{C}_{\mathtt{B}} \cap \mathcal{P}_{\mathtt{sid}}}$ are interpreted as the messages sent by corrupted parties in the previous round. On the RN-th input, where $\mathsf{RN} < N + 1$, it returns: (1) the RN-th-round messages generated by honest parties and sent to parties in $\mathcal{C}_{\mathtt{B}}$, and (2) the complete views of the parties in $\mathcal{C}_{\mathtt{P}}$, including the messages they received from honest parties in that round, the messages they sent in that round, and their internal states. On the $(N + 1)$-th input, it returns $(\mathtt{sid}, \mathtt{final}, pk, (pk_j)_{j \in [n_{\mathtt{sid}}]})$, where $pk = y$.

Definition 2 (Concurrent Full Secrecy)**.** An Anchor-DKG scheme satisfies concurrent full secrecy if there exists a polynomial-time simulator Sim such that, for any PPT adversary $\mathcal{A}$ and any $\lambda \in \mathbb{N}$, it holds that

$$\Pr\left[\begin{array}{l} b \leftarrow\!\!\$\ \{0,1\}; y \leftarrow\!\!\$\ \mathbb{G}; \mathcal{U}, \mathcal{H}, \mathcal{C}_{\mathtt{B}}, \mathcal{C}_{\mathtt{P}}, \mathcal{MSK}, \mathsf{InsIDs} \leftarrow \emptyset; \\ b' \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{AddU}}, \mathcal{O}_{\mathsf{newIns}}, \mathcal{O}_b(y,\cdot), \mathsf{RO}_b}(1^\lambda) : b = b' \end{array}\right] \leq \frac{1}{2} + \mathsf{negl}(\lambda). \tag{5}$$

Here, $\mathcal{O}_1$ is the oracle $\mathcal{O}_{\mathsf{Chall}}$ defined above, ignoring the input y. The oracle $\mathcal{O}_0$ is a simulation oracle with both read and write access to InsIDs, and serves as a relay between the adversary and the simulator Sim. In particular, when $\mathcal{O}_0$ receives from $\mathcal{A}$ a valid instance-initialization message $(\mathtt{sid}, \mathtt{init}, \mathcal{R}_{\mathsf{init}}, \mathcal{N}_{\mathsf{sid}}, t^{\mathsf{rec}}_{\mathtt{sid}})$ such that $\mathtt{sid} \notin \mathsf{InsIDs}$, it invokes Sim with the following initial inputs: the group element $y \in \mathbb{G}$, generators $g, h \in \mathbb{G}$, the instance information $(\mathtt{sid}, \mathtt{init}, \mathcal{R}_{\mathsf{init}}, \mathcal{N}_{\mathsf{sid}}, t^{\mathsf{rec}}_{\mathtt{sid}})$, the corruption information $(\mathcal{C}_{\mathtt{B}}, \mathcal{C}_{\mathtt{P}})$, and the master keys of all honest-but-curious parties $\{msk_i\}_{i \in \mathcal{C}_{\mathtt{P}}}$. Thereafter, $\mathcal{O}_0$ simply forwards execution messages from the adversary to Sim and returns the simulator's outputs to the adversary. Note that the final message from Sim must be in the form of $(\mathtt{sid}, \mathtt{final}, y, \dots)$.

The oracle RO_1 consists of the real random oracles, whereas RO_0 is managed by Sim. Furthermore, $\mathcal{A}$ may query $\mathcal{O}_{\mathsf{newIns}}$ with polynomially many distinct instance-initialization messages, and may query $\mathcal{O}_b$ with at most one instance-initialization message.

Remark 3. While the above definition is tailored for random-oracle-model constructions, a general definition can be obtained by removing the adversary's access to RO_b. Looking ahead, our Anchor-DKG scheme relies on the random oracle only because of the underlying NIZK.

On composition security with subsequent threshold schemes. Our definitions ensure not only the concurrent security of the Anchor-DKG protocol, but also that the subsequent threshold scheme is secure when it is arbitrarily composed with threshold schemes that use keys from *other* Anchor-DKG instances. Specifically, as the oracle $\mathcal{O}_{\mathsf{newIns}}$ already reveals all honest secret key shares of non-challenging instances to the adversary, any messages generated by threshold schemes using those keys can be perfectly simulated and thus leak no additional information. Security under concurrent use of threshold schemes instantiated *from the same challenge instance* depends on the threshold scheme itself. Nevertheless, because our protocol satisfies full secrecy, it can safely instantiate a wide range of concurrently secure threshold schemes, such as [19] [42].

4 Our Anchor-DKG Protocol

4.1 Ingredients

Our protocol utilizes the following tools.

Pedersen VSS. We employ a variant of Pedersen VSS in a *non-black-box* manner. Let $\mathbb{G}$ be a cyclic group with a prime order q, and let g and h be two independent generators of $\mathbb{G}$. To verifiably share a secret $s \in \mathbb{Z}_q$ with a reconstruction threshold t^{rec}, a dealer first uniformly samples two $t^{\mathsf{rec}}-1$-degree polynomials $f(x), f'(x) \in \mathbb{Z}_q[X]$ under the condition that $f(0) = s$. Then, it generates the following commitment to $f(x)$ where $f'(x)$ serves as the hiding factor:$\mathsf{PC} = (g^{f(j)}h^{f'(j)})_{j\in[0,n]}$. Next, it broadcasts PC and sends $(f(i), f'(i))$ to each receiver P_i. After receiving (s_i, s_i'), P_i can verify whether s_i is an evaluation of the committed polynomial at i by performing the following operations.

- **Degree Check.** First, P_i invokes $\mathsf{DegCheck}(\mathsf{PC}, n, t^{\mathsf{rec}}-1)$ to check whether PC commits to a $(t^{\mathsf{rec}}-1)$-degree polynomial in $\mathbb{Z}_q[X]$. $\mathsf{DegCheck}$ is a randomized algorithm introduced in [17] and employed in many subsequent DKG designs[28, 9, 29, 20]. Except with a negligible probability, this algorithm returns 1, iff there exist $(t^{\mathsf{rec}}-1)$-degree $f(x), f'(x) \in \mathbb{Z}_q[X]$, such that $\mathsf{PC} = (g^{f(i)}h^{f'(i)})_{i\in[0,n]}$.

- **Evaluation Check.** Then, P_i simply verifies whether $g^{s_i}h^{s_i'}$ equals to the $i+1$-th element of PC.

We use commitments to evaluations rather than coefficients because it reduces the computational cost of verifying $O(n)$ shares against the same commitment, as the degree check only needs to be performed once

NIZKPoK. In the random oracle model, a NIZKPoK for an NP relation Rela can be described with two algorithms, Prove and Vrfy, where the former algorithm on inputs a statement X and a witness W such that $(X, W) \in \mathsf{Rela}$ generates a proof π, and the latter verifies (X, π). A NIZKPoK shall satisfy *completeness*, *zero-knowledge*, and *knowledge soundness* properties, which we recall in Appendix.A.

In our Anchor-DKG, we utilize the following two NIZKPoK schemes for different relations.

- $\mathsf{NIZK}_{\mathsf{Ped}}$ for the relation

$$\mathsf{Rela}_{\mathsf{Ped}} := \{(X = (C, g, h, s), W = s') : C = g^s h^{s'}\}, \tag{6}$$

which is used to provide a simulatable opening for a Pedersen commitment C, as we discussed in §2.2.

- $\mathsf{NIZK}_{\mathsf{pk}}$ for the relation $\mathsf{Rela}_{\mathsf{pk}} :=$

$$\{(X = (C, g, h, pk), W = (s, s')) : C = g^s h^{s'} \wedge pk = g^s\}, \tag{7}$$

  which demonstrates a public key (share) pk is honestly computed.

We may omit the public generators g and h in statements when there is no ambiguity. Both $\mathsf{NIZK}_{\mathsf{Ped}}$ and $\mathsf{NIZK}_{\mathsf{pk}}$ constructions are standard, and we include them in Appendix A for completeness.

Pseudorandom Function. In the last, we use a conventional PRF PRF as our key derivation function, which can be described by the following algorithms: $\mathsf{PRF.KeyGen}(1^\lambda)$. It generates a PRF key k; $\mathsf{PRF.Eval}(k, \mathtt{sid})$. The evaluation algorithm takes a key and a string $\mathtt{sid}$ in the space of session IDs as inputs and outputs $s \in \mathbb{Z}_p$.

4.2 Protocol Description

The master key generation algorithm MkGen and the share derivation algorithm $\mathsf{ShareDeriv}$ of our $\mathsf{Anchor\text{-}DKG}$ simply invoke PRF.
KeyGen and $\mathsf{PRF.Eval}$, respectively.

Regarding the main protocol of our $\mathsf{Anchor\text{-}DKG}$, we present the pseudocode in Fig.2 [8]. For notational simplicity, we use variables such as n, t^{rec}, and pk, instead of $n_{\mathtt{sid}}, t^{\mathsf{rec}}_{\mathtt{sid}}, pk[\mathtt{sid}]$, in the description.

In particular, the protocol execution generally consists of the following two phases.

First Phase: At a high level, in the first phase, every party verifiably shares a secret with the network, and the network agrees on a set $\widetilde{R}_{\mathtt{sid}}$ of t^{rec} qualified VSS instances. Specifically, each repeating party P_i shares its derived key $\mathsf{ShareDeriv}(msk_i, \mathtt{sid})$, while non-repeating parties simply share independently sampled random secrets. This phase spans the first three rounds and concludes at Line 14 of Round 4.

- **Round 1**: Each P_i determines its secret s_i to be shared based on its role (Lines 1–3). It then acts as a dealer in Pedersen VSS to share s_i. Specifically, it samples a secret-sharing polynomial $\widehat{f}_i$ such that $\widehat{f}_i(0) = s_i$, and a hiding polynomial $\widehat{f}'_i$ (Lines 4–5), then generates a polynomial commitment PC_i (Line 6). The commitment PC_i is broadcast to the network, while the pair $(\widehat{f}_i(j), \widehat{f}'_i(j))$ is sent privately to each P_j.

- **Rounds 2 and 3**: These are standard complaint rounds. Each P_i performs $\mathsf{DegCheck}$ on each received PC_j; if the check fails, P_j is immediately disqualified (Lines 6–7). Otherwise, if the share received from P_j does not match the commitment PC_j (or no share is received), P_i adds P_j to its complaint list (Lines 8–9). Valid shares are stored locally (Line 10), and the complaint list is broadcast (Line 11). In Round 3, each P_i responds to complaints against itself by broadcasting the relevant share to the complainant P_j.

[8] To provide a more intuitive understanding, we illustrate the actions of each P_i in different rounds in Fig.2, which can be directly translated into the round and output functions used in our formal modeling.

Round 1: each P_i **do**:

$P_i \in \mathcal{R}_{\mathtt{sid}}$ **then** // A repeating party
  Derive $s_i \leftarrow \mathsf{ShareDeriv}(msk_i, \mathtt{sid})$
else Sample $s_i \leftarrow\!\!\$\ \mathbb{Z}_p$ // A non-repeating party
Sample $\widehat{f}_i, \widehat{f}'_i \leftarrow\!\!\$\ \mathbb{Z}_p[X]$, s.t.,
  $\mathsf{deg}(\widehat{f}_i) = \mathsf{deg}(\widehat{f}'_i) = t^{\mathsf{rec}} - 1, \widehat{f}_i(0) = s_i$
Commit: $\mathsf{PC}_i \leftarrow (g^{\widehat{f}_i(0)} h^{\widehat{f}'_i(0)}, \ldots, g^{\widehat{f}_i(n)} h^{\widehat{f}'_i(n)})$
Broadcast $(\mathtt{sid}, \mathsf{PC}_i)$
Send $(\mathtt{sid}, i, j, \widehat{s}_{i,j} = \widehat{f}_i(j), \widehat{s}'_{i,j} = \widehat{f}'_i(j))$
  to each $P_j \in \mathcal{P}_{\mathtt{sid}}$

Round 2: each P_i **do**:

Receive: $\{\mathtt{sid}, \mathsf{PC}_j\}_{j\in[n]}$
  and $\{(\mathtt{sid}, j, i, \widehat{s}_{j,i}, \widehat{s}'_{j,i})\}_{j\in[n]}$
Parse and Store $\mathsf{PC}_j = (\widehat{A}_{j,0}, \ldots, \widehat{A}_{j,n}), \forall j \in [n]$
Set $\mathsf{DisQual}, \mathsf{Complaint}_i, \mathsf{Shares}_i \leftarrow \emptyset$
for $j \in [n]$ **do**
  if $\mathsf{DegCheck}(\mathsf{PC}_j, n, t) = 0$, **then**
    $\mathsf{DisQual} \leftarrow \mathsf{DisQual} \cup \{j\}$
  elseif $g^{\widehat{s}_{j,i}} h^{\widehat{s}'_{j,i}} \neq \widehat{A}_{j,i}$ **then**
    $\mathsf{Complaint}_i \leftarrow \mathsf{Complaint}_i \cup \{(i, j)\}$
  else $\mathsf{Shares}_i \leftarrow \mathsf{Shares}_i \cup \{(j, i, \widehat{s}_{j,i}, \widehat{s}'_{j,i})\}$
Broadcast $(\mathtt{sid}, \mathsf{Complaint}_i)$

Round 3: each P_i **do**:

Receive:$\{(\mathtt{sid}, \mathsf{Complaint}_j)\}_{j\in[n]}$
Set $\mathsf{Complaint} \leftarrow \emptyset, \mathsf{Resp}_i \leftarrow \emptyset$
for $j \in [n]$ **do**
  if $\neg\exists j' \neq j \wedge (j', \cdot) \in \mathsf{Complaint}_j$ **then**
    $\mathsf{Complaint} \leftarrow \mathsf{Complaint} \cup \mathsf{Complaint}_j$
for $(j, i) \in \mathsf{Complaint}$ **do**
  $\mathsf{Resp}_i \leftarrow \mathsf{Resp}_i \cup \{(j, i, \widehat{s}_{i,j}, \widehat{s}'_{i,j})\}$
Broadcast $(\mathtt{sid}, \mathsf{Resp}_i)$

At the end of Round 5: each P_i **do**:

Receive $\{(\mathtt{sid}, \mathsf{Open}_j = \{(\widehat{s}_{z,j}, \widehat{s}'_{z,j})\}_{z\in\widetilde{\mathsf{DisQual}}})\}_{j\in[n]}$
for $z \in \widetilde{\mathsf{DisQual}}$, Set $\mathsf{ValidOpen}_z \leftarrow \emptyset$
  for $j \in [n]$, **if** $g^{\widehat{s}_{z,j}} h^{\widehat{s}'_{z,j}} = \widehat{A}_{z,j}$
    $\mathsf{ValidOpen}_z \leftarrow \mathsf{ValidOpen}_z \cup \{(j, \widehat{s}_{z,j})\}$
    if $|\mathsf{ValidOpen}_z| = t^{\mathsf{rec}}$ **then**
      Interpolate sk_z from $\mathsf{ValidOpen}_z$
      $pk_z \leftarrow g^{sk_z}$
Interpolate pk, and $\{pk_z\}_{z\in[n]\setminus\widetilde{R}_{\mathtt{sid}}}$, using $\{pk_z\}_{z\in\widetilde{R}_{\mathtt{sid}}}$
return $(pk, (pk_z)_{z\in[n]}, sk_i)$

Round 4: each P_i **do** :

// Decide the qualified set of VSS
receive $\{(\mathtt{sid}, \mathsf{Resp}_j)\}_{j\in[n]}$
for $j \in [n] \setminus \mathsf{DisQual}$ **do**
  if $\exists(i', j) \in \mathsf{Complaint} \wedge \neg\exists(i', j, \widehat{s}_{j,i'}, \widehat{s}'_{j,i'}) \in \mathsf{Resp}_j$
    s.t., $\widehat{A}_{j,i'} = g^{\widehat{s}_{j,i'}} h^{\widehat{s}'_{j,i'}}$
  then $\mathsf{DisQual} \leftarrow \mathsf{DisQual} \cup \{j\}$
$\mathsf{Qual} \leftarrow [n] \setminus \mathsf{DisQual}$
// Store all valid shares from qualified dealers
$\mathsf{Share}_i \leftarrow \mathsf{Share}_i \cup \{(j, i, \widehat{s}_{j,i}, \widehat{s}'_{j,i}) \in \mathsf{Resp}_j\}_{j\in\mathsf{Qual}}$
if $u = t^{\mathsf{rec}} - |\mathcal{R}_{\mathtt{sid}} \cap \mathsf{Qual}| > 0$ **then**
  Let $i_1, \ldots, i_u \in \mathsf{Qual} \setminus \mathcal{R}_{\mathtt{sid}}$ be the smallest u indexes
  $\widetilde{R}_{\mathtt{sid}} = \mathcal{R}_{\mathtt{sid}} \cap \mathsf{Qual} \cup \{i_k\}_{k\in[u]}$
// $\widetilde{R}_{\mathtt{sid}}$ consists of t^{rec} qualified parties including $\mathcal{R}_{\mathtt{sid}}$
else $\widetilde{R}_{\mathtt{sid}} = \mathcal{R}_{\mathtt{sid}}$
Interpolate $f_i, f'_i \in \mathbb{Z}_q[X], \mathsf{deg}(f_i) = t^{\mathsf{rec}} - 1$, s.t.
  $f_i(j) = \widehat{s}_{j,i}, f'_i(j) = \widehat{s}'_{j,i}, \forall j \in \widetilde{R}_{\mathtt{sid}}$
for $j \in [n] \setminus \widetilde{R}_{\mathtt{sid}}, \pi_{i,j} \leftarrow \mathsf{NIZK}_{\mathsf{Ped}}(g^{f_i(j)} h^{f'_i(j)}, f_i(j), f'_i(j))$
Send $(\mathtt{sid}, (i, j, s_{i,j} = f_i(j), \pi_{i,j}))$ to each $P_j \in [n] \setminus \widetilde{R}_{\mathtt{sid}}$
if $P_i \in \widetilde{R}_{\mathtt{sid}}$ **then** $sk_i \leftarrow s_i$
  Compute:$pk_i \leftarrow g^{s_i}, \gamma_i \leftarrow \mathsf{NIZK}_{\mathsf{pk}}(\widehat{A}_{i,0}, pk_i, (s_i, \widehat{f}'_i(0)))$
  Broadcast $(\mathtt{sid}, pk_i, \gamma_i)$

Round 5 or End of Round 4: each P_i **do** :

Receive $\{(\mathtt{sid}, (j, i, s_{j,i}, \pi_{i,j}))\}_{j\in[n]}, \{(\mathtt{sid}, pk_j, \gamma_j)\}_{j\in\widetilde{R}_{\mathtt{sid}}}$
Set $\mathcal{PK}, \widetilde{\mathsf{DisQual}} \leftarrow \emptyset$
for $j \in \widetilde{\mathcal{R}}_{\mathtt{sid}}$, **if** $\mathsf{NIZK}_{\mathsf{pk}}.\mathsf{Vrfy}(\widehat{A}_{j,0}, pk_j, \gamma_j) = 1$ **then**
  $\mathcal{PK} \leftarrow \mathcal{PK} \cup \{pk_j\}$
  else $\widetilde{\mathsf{DisQual}} \leftarrow \widetilde{\mathsf{DisQual}} \cup \{j\}$
if $P_i \notin \widetilde{R}_{\mathtt{sid}}$, **then** Set $\mathsf{ValidShares} \leftarrow \emptyset$
  for $j \in [n]$ **do** // $\widehat{\lambda}_\tau$ defined in Eq.8
    $A_{j,i} = \prod_{\tau\in\widetilde{R}_{\mathtt{sid}}} \widehat{A}_{\tau,j}^{\widehat{\lambda}_\tau(i)}$ // Reconstruct $g^{f_j(i)} h^{f_j(i)}$
    if $\mathsf{NIZK}_{\mathsf{Ped}}.\mathsf{Vrfy}(A_{j,i}, s_{j,i}, \pi_{i,j}) = 1$ **then**
      $\mathsf{ValidShares} \leftarrow \mathsf{ValidShares} \cup \{j, s_{j,i}\}$
      if $|\mathsf{ValidShares}| = t^{\mathsf{rec}}$ **then**
        $sk_i \leftarrow \sum_{\zeta\in\mathsf{ValidShares}} \lambda_\zeta(0) s_{\zeta,i}$ // λ_ζ defined in Eq.9
if $\widetilde{\mathsf{DisQual}} = \emptyset$ **then**
  Interpolate pk, and $\{pk_z\}_{z\in[n]\setminus\widetilde{R}_{\mathtt{sid}}}$, using $\{pk_z\}_{z\in\widetilde{R}_{\mathtt{sid}}}$
  return $(pk, (pk_z)_{z\in[n]}, sk_i)$ // Protocol ends
else Multicast $(\mathtt{sid}, \mathsf{Open}_i = \{(\widehat{s}_{j,i}, \widehat{s}'_{j,i})\}_{j\in\widetilde{\mathsf{DisQual}}})$

Figure 2: Our Anchor-DKG protocol. For notational simplicity, we omit `sid` in the representation of most variables.

- **Round 4 (up to Line 15)**: The network determines the set Qual of qualified dealers—those who received no complaints or successfully responded to all complaints (Lines 1–7). From Qual, a deterministic rule selects t^{rec} parties to form the set $\widetilde{R}_{\mathtt{sid}}$, ensuring that all repeating parties in Qual are included.

<u>*Second Phase:*</u> At the beginning of the second phase, all parties in $\widetilde{R}_{\mathtt{sid}}$ have already known their secret key shares, and thus the public key has also been determined (implicitly). The remaining tasks for this phase include (1) to help parties not in $\widetilde{R}_{\mathtt{sid}}$ obtain their secret key shares, and (2) to obtain the public key pk and public key shares $(pk_i)_{i\in[n]}$.

- **Round 4 after Line 15:** For helping the parties not in $\widetilde{R}_{\mathtt{sid}}$, each party P_i interpolates $(t^{\mathsf{rec}}-1)$-degree polynomials f_i, such that $f_i(j) = \widehat{s}_{j,i} = \widehat{f}_j(i)$, $f'_i(j) = \widehat{s}'_{j,i} = \widehat{f}'_j(i)$ for all $j \in \widetilde{R}_{\mathtt{sid}}$ (Line 15-16). To each $P_j \notin \widetilde{R}_{\mathtt{sid}}$, P_i sends the evaluation $f_i(j)$ along with a NIZKPoK proof $\pi_{i,j}$ which demonstrates the knowledge of $f'_i(j)$ (Line 17-18). Note that $(f_i(j), \pi_{i,j})$ can serve as an opening of $g^{f_i(j)}h^{f'_i(j)}$.

  For generating the public key (shares), each repeating party P_i broadcasts its public key share $pk_i = g^{s_i}$ along with a NIZKPoK proof ζ_i that demonstrates that pk_i is honestly computed from the secret committed by $A_{i,0}$ (Line 19-21).

- **End of Round 4 or Round 5:** To obtain the secret key share sk_i, a party P_i not in $\widetilde{R}_{\mathtt{sid}}$ receives the pairs $\{(s_{j,i}, \pi_{j,i})\}_{j\in[n]}$ from other P_j's (Line 1). For verifying whether $s_{j,i}$ is an honest evaluation of the polynomial f_j interpolated by P_j, P_i reconstructs the commitment $g^{f_j(i)}h^{f'_j(i)}$ as $\prod_{\tau\in\widetilde{R}_{\mathtt{sid}}} \widehat{A}_{\tau,j}^{\lambda_\tau(i)}$ (Line 9), where

$$\widehat{\lambda}_\tau(i) = \prod_{m\in\widetilde{R}_{\mathtt{sid}}, m\neq\tau} \frac{i-m}{\tau-m} \tag{8}$$

  Then, P_i verifies the proof $\pi_{j,i}$ against $g^{f_j(i)}h^{f'_j(i)}$ and $s_{j,i}$ (Line 10). If the verification passes, $s_{j,i}$ is recorded as a valid share in $\mathsf{ValidShares}$ (Line 11). When there are t^{rec} shares in $\mathsf{ValidShares}$, sk_i can be computed as $\sum_{\zeta\in\mathsf{ValidShares}} \widehat{\lambda}_\zeta(0)\widehat{s}_{\zeta,i}$ (Line 12-14), where

$$\lambda_\zeta(j) = \prod_{(m,\cdot)\in\mathsf{ValidShares}, m\neq\zeta} \frac{j-m}{\zeta-m} \tag{9}$$

  To generate the public key, each P_i verifies pk_j provided by every P_j in $\widetilde{R}_{\mathtt{sid}}$ (Line 4-5). If some $P_j \in \widetilde{R}_{\mathtt{sid}}$ does not provide a valid pk_j, it will be disqualified and recorded in $\widetilde{\mathsf{DisQual}}$ (Line 6).

  If $\widetilde{\mathsf{DisQual}}$ is empty, which means there are t^{rec} valid public key shares in $\mathcal{PK}$, P_i can interpolate these public key shares to obtain the group public key pk and other public key shares (Line 14-16), and the protocol can end here. Otherwise, P_i multicasts shares received from parties in $\widetilde{\mathsf{DisQual}}$ (Line 15).

- **The end of Round 5:** Each P_i can receive at least t^{rec} shares delivered by any $P_j \in \widetilde{\mathsf{DisQual}}$ (Line 2-5). Then, the secret key $sk_z = \widehat{f}_z(0)$ can be interpolated, which gives the public key share $pk_z = g^{sk_z}$ (Line 7). Finally, the public key pk and other public key shares for parties not in $\widetilde{R}_{\mathtt{sid}}$ can be obtained by interpolating $\{pk_z\}_{z\in\widetilde{R}_{\mathtt{sid}}}$.

Correctness. As a sanity check, we show that our protocol can function as expected when there is no corrupted parties, i.e., (1) repeating parties output their derived keys as their secret key shares, and that (2) the secret key shares outputted by honest parties are on a $(t^{\mathsf{rec}}-1)$-degree polynomial F, and $sk = F(0)$.

For (1), a repeating P_i will be included in Qual and appear in $\widetilde{R}_{\mathtt{sid}}$. According to our protocol, each P_i in $\widetilde{R}_{\mathtt{sid}}$ directly outputs s_i as its secret key share.

For (2), the final polynomial F is defined by the secrets of parties in $\widetilde{R}_{\mathtt{sid}}$, i.e., $F(x) = \sum_{i\in\widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_i(x)s_i$. For $P_i \notin \widetilde{R}_{\mathtt{sid}}$, according to our protocol, its secret key share is computed as follows.

$$\begin{aligned} sk_i &= \sum_{\zeta\in\mathsf{ValidShares}} \lambda_\zeta(0)s_{\zeta,i} = \sum_{\zeta\in\mathsf{ValidShares}} \lambda_\zeta(0)f_\zeta(i) \\ &= \sum_{\zeta\in\mathsf{ValidShares}} \lambda_\zeta(0)\Big(\sum_{\tau\in\widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i)\widehat{f}_\tau(\zeta)\Big) \\ &= \sum_{\tau\in\widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i)\Big(\sum_{\zeta\in\mathsf{ValidShares}} \lambda_\zeta(0)\widehat{f}_\tau(\zeta)\Big) \\ &= \sum_{\tau\in\widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i)\widehat{f}_\tau(0) = \sum_{\tau\in\widetilde{R}_{\mathtt{sid}}} \lambda_\tau(i)F(\tau) = F(i), \end{aligned} \tag{10}$$

where the second line follows how f_ζ is constructed by interpolating $f_\zeta(i) = \hat{f}_i(\zeta)$ for all $i \in \widetilde{R}_{\mathtt{sid}}$.

Variants with fewer rounds. We remark that by employing the publicly verifiable secret sharing (PVSS) scheme with Pedersen commitment [20], the round complexity of the protocol in Fig. 2 can be trivially reduced to 2 broadcast rounds (plus one or two point-to-point rounds, depending on the variant of [20]), as the complaint phase (Rounds 2 and 3) is no longer needed. Nonetheless, using PVSS introduces significant computational overhead (due to the need to verify encrypted shares) and requires a PKI setup.

4.3 Security Analysis

We establish the following theorem.

Theorem 1. *Under the discrete logarithm assumption, the protocol in Fig.2 satisfies the concurrent robustness (cf. Def.1)and full secrecy (cf. Def. 2) in the random oracle model.*

Proof outline. We outline the core steps below.

First, to simplify the concurrent security analysis, we introduce the PRF-oracle-enhanced security notion in Def. 3, where the adversary may additionally query a PRF oracle whose internal state consists of the master keys of honest parties. In Theorem 2, we show that $\mathcal{O}_{\mathsf{newIns}}$ can be perfectly emulated using this PRF oracle, so it suffices to prove PRF-oracle-enhanced security.

Next, in Lemma 3, we establish robustness. The soundness of the underlying NIZKs and the binding property of the Pedersen commitments (both implied by the discrete logarithm assumption) ensure that malicious participants cannot send inconsistent shares without detection, allowing the rest of the argument to follow the correctness analysis.

Finally, Lemma 4 proves full secrecy by constructing a PPT simulator Sim. While most of the simulation is standard, the simulation of Round 4 messages is unique to our protocol, leveraging the perfectly hiding property and the simulator of the NIZKs.

Simplifying concurrent security analysis. Our modeling captures the adversary's ability to create and participate in other instances by granting it access to the oracle $\mathcal{O}_{\mathsf{newIns}}$. However, since this oracle essentially provides all protocol messages in the created instances to the adversary, directly analyzing the impact of each message on the security of the challenge instance can be cumbersome.

[30] recently introduced a framework for simplifying concurrent security analysis. At a high level, in their framework, a protocol instance Π_1 is said to be $\mathbb{O}$-*emulatable* if all messages from Π_1 can be *perfectly* simulated by a PPT algorithm given access to oracles in the set $\mathbb{O}$. Their composition lemma then states that another instance Π_2 remains secure when executed concurrently with Π_1, provided that Π_2 is secure when the adversary has oracle access to $\mathbb{O}$.

Following their proof strategy, we observe that instances of Anchor-DKG can be emulated using a PRF oracle that, on input of a node ID i and a session ID $\mathtt{sid}$, returns $\mathsf{PRF}(msk_i, \mathtt{sid})$. Consequently, we only need to show that our protocol remains secure even when the adversary is given access to this PRF oracle.

Formally, we define the security of Anchor-DKG against oracle-enhanced adversaries as follows.

Definition 3 (PRF-oracle-enhanced Security)**.** Let $\mathcal{O}_{\mathsf{PRF}}$ be an oracle, which has the master keys of all honest parties $\mathcal{MSK}$, as its internal state, and has writing access to the set of instance IDs InsIDs. On input $(\mathtt{sid}, i)$ for some $i \in \mathcal{H}$, $\mathcal{O}_{\mathsf{PRF}}$ performs as follows:

- Ignore this input if $\mathtt{sid} \in$ InsIDs; Otherwis, add $\mathtt{sid}$ to InsIDs;
- Return $\mathsf{PRF.Eval}(msk_i, \mathtt{sid})$.

The defintions for robustness and full secrecy against $\mathcal{O}_{\mathsf{PRF}}$-enhanced adversaries are almost identical to the concurrent robustness and full secrecy definitions as in Def.1 Def.2, respectively, with the exception that the adversary here has access to $\mathcal{O}_{\mathsf{PRF}}$ rather than $\mathcal{O}_{\mathsf{newIns}}$.

For our protocol in Fig.2, security against oracle-enhanced adversaries implies the concurrent security.

Theorem 2. *If* Anchor-DKG *in Fig.2 satisfies the* $\mathcal{O}_{\mathsf{PRF}}$*-enhanced security as per Def.3, then it satisfies the concurrent security as per Def.1 and Def.2.*

Proof. Assume there exists a PPT adversary $\mathcal{A}$ that violates the concurrent security (either full secrecy or robustness) of our Anchor-DKG protocol. Then, there exists a PPT adversary $\mathcal{B}$ that breaks the $\mathcal{O}_{\mathsf{PRF}}$-enhanced security of Anchor-DKG by invoking $\mathcal{A}$ as a subroutine.

The adversary $\mathcal{B}$ simulates the concurrent environment for $\mathcal{A}$ using access to the PRF oracle $\mathcal{O}_{\mathsf{PRF}}$. Specifically, whenever $\mathcal{A}$ makes a query to $\mathcal{O}_{\mathsf{newIns}}$ with parameters $(\mathtt{sid}, \mathtt{init}, \mathcal{R}_{\mathtt{sid}}, \mathcal{N}_{\mathtt{sid}}, t^{\mathsf{rec}}_{\mathtt{sid}}, \vec{T}_{\mathtt{sid}})$, $\mathcal{B}$ proceeds as follows to perfectly simulate the resulting instance:

- For each $i \in \mathcal{R}_{\mathtt{sid}} \cap \mathcal{H}$, query $\mathcal{O}_{\mathsf{PRF}}$ with $(\mathtt{sid}, i)$ and obtain the derived secret $s_i[\mathtt{sid}]$.
- Run the internal logic of $\mathcal{O}_{\mathsf{newIns}}$ honestly to simulate all subsequent queries involving session $\mathtt{sid}$, with one exception: when simulating the first-round behavior of honest repeating parties (i.e., executing $\mathsf{Round}(\mathtt{sid}, i, 1, \dots)$ for $i \in \mathcal{R}_{\mathtt{sid}} \cap \mathcal{H}$), $\mathcal{B}$ skips Line 2 of Round 1 in Fig. 2, and instead directly sets $s_i[\mathtt{sid}]$ to the value returned by $\mathcal{O}_{\mathsf{PRF}}$.

By construction, the view of $\mathcal{A}$ in this simulation is identically distributed to the real execution. Thus, if $\mathcal{A}$ succeeds in breaking the concurrent security of Anchor-DKG, then $\mathcal{B}$ succeeds in breaking the $\mathcal{O}_{\mathsf{PRF}}$-enhanced security. □

Remark 4. Theorem 2 demonstrates that we only need to focus on a standalone adversary with oracle access to $\mathcal{O}_{\mathsf{PRF}}$, which allows us to apply tools that are not available in a fully concurrent setting, such as the knowledge extractor of the underlying NIZKPOK.

Robustness. We now prove the concurrent robustness of our Anchor-DKG protocol by showing that it satisfies $\mathcal{O}_{\mathsf{PRF}}$-enhanced robustness.

Lemma 3 (Robustness)**.** *The* Anchor-DKG *protocol in Fig. 2 satisfies $\mathcal{O}_{\mathsf{PRF}}$-enhanced robustness. Consequently, it also satisfies concurrent robustness as defined in Definition 1.*

Proof. Recall that Anchor-DKG satisfies robustness if the challenge oracle $\mathcal{O}_{\mathsf{Chall}}$ does not return a `Failed-Instance` message, except with negligible probability. According to Fig. 1, we must ensure that, in the challenge instance—where the number of corrupted parties does not exceed the threshold—the following conditions hold:

- Honest repeating parties output secret key shares identical to their derived keys (Line 33).
- All secret key shares lie on the same $(t^{\mathsf{rec}} - 1)$-degree polynomial F, and the group public key satisfies $pk = g^{F(0)}$ (Lines 34–35).
- All honest parties output the same pk and the same vector of public key shares $(pk_i)_{i\in[n]}$ (Line 32 of Fig. 1).

For the first condition, according to our protocol, all parties in $\widetilde{R}_{\mathtt{sid}}$ will output s_i as their secret key share. A repeating party is included in $\widetilde{R}_{\mathtt{sid}}$ whenever it is part of the set Qual. It is straightforward to verify that all honest parties are included in Qual, and hence every honest repeating party will be part of $\widetilde{R}_{\mathtt{sid}}$. Therefore, each honest repeating party outputs $s_i = \mathsf{ShareDeriv}(msk_i, \mathtt{sid})$ as its secret key share, satisfying the requirement.

For the second requirement, recall that the final polynomial F is defined using the secrets of parties in $\widetilde{R}_{\mathtt{sid}}$, i.e.,

$$F(x) = \sum_{i\in\widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_i(x) s_i.$$

For each $P_i \notin \widetilde{R}_{\mathtt{sid}}$, our protocol computes its secret key share as:

$$sk_i = \sum_{\zeta\in\mathsf{ValidShares}} \lambda_\zeta(0) s_{\zeta,i}, \tag{11}$$

where for every $\zeta \in \mathsf{ValidShares}$, there exists a valid NIZKPoK proof $\pi_{\zeta,i}$ demonstrating knowledge of $s'_{\zeta,i}$ such that

$$g^{s_{\zeta,i}} h^{s'_{\zeta,i}} = A_{\zeta,i} = \prod_{\tau\in\widetilde{R}_{\mathtt{sid}}} \widehat{A}_{\tau,\zeta}^{\widehat{\lambda}_\tau(i)}.$$

By the knowledge soundness of $\mathsf{NIZK}_{\mathsf{Ped}}$, an extractor can recover $s_{\zeta,i}$ and $s'_{\zeta,i}$ from the adversary such that the above relation holds.

Note that for each $(\widehat{A}_{i,0}, \ldots, \widehat{A}_{i,n})$ with $i \in \widetilde{R}_{\mathtt{sid}}$, there exist $(t^{\mathsf{rec}}-1)$-degree polynomials $\widehat{f}_i$ and $\widehat{f}'_i$ (which can be reconstructed from shares received by honest or passively corrupted parties even if P_i is Byzantine corrupted) satisfying:

$$\widehat{A}_{i,j} = g^{\widehat{f}_i(j)} h^{\widehat{f}'_i(j)}.$$

Define

$$f_\zeta(x) = \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(x) \widehat{f}_\tau(\zeta), \quad f'_\zeta(x) = \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(x) \widehat{f}'_\tau(\zeta),$$

then we have

$$A_{\zeta,i} = g^{f_\zeta(i)} h^{f'_\zeta(i)}.$$

By the binding property of Pedersen commitments (e.g., under the discrete log assumption), this implies

$$s_{\zeta,i} = f_\zeta(i) = \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i) \widehat{f}_\tau(\zeta).$$

Substituting this into Equation (11), we obtain:

$$\begin{aligned} sk_i &= \sum_{\zeta \in \mathsf{ValidShares}} \lambda_\zeta(0) \left(\sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i) \widehat{f}_\tau(\zeta) \right) \\ &= \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i) \left(\sum_{\zeta \in \mathsf{ValidShares}} \lambda_\zeta(0) \widehat{f}_\tau(\zeta) \right) \\ &= \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i) \widehat{f}_\tau(0) = \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{\lambda}_\tau(i) F(\tau) = F(i), \end{aligned} \tag{12}$$

which shows that all secret key shares output by honest parties lie on the polynomial F.

Moreover, by the completeness of $\mathsf{NIZK}_{\mathsf{Ped}}$, all valid shares from honest or passively corrupted parties will be accepted into $\mathsf{ValidShares}$, ensuring that every honest party has enough valid points to compute its share.

For the third requirement, note that all sk_z values reconstructed at the end of Round 5 must equal the values s_z shared in the first round by P_z, which by definition satisfies $s_z = F(z)$. For any other $pk_i \in \mathcal{PK}$, there exists a valid NIZKPoK proof γ_i that demonstrates knowledge of (s_i, s'_i) such that

$$\widehat{A}_{i,0} = g^{s_i} h^{s'_i} \quad \text{and} \quad pk_i = g^{s_i}.$$

By the knowledge soundness of $\mathsf{NIZK}_{\mathsf{pk}}$ and the binding property of Pedersen commitments, it follows that $pk_i = g^{F(i)}$.

Furthermore, every honest party can obtain the set $\{pk_i = g^{F(i)}\}_{i \in \widetilde{R}_{\mathtt{sid}}}$, because all honest or passively corrupted parties in $\widetilde{R}_{\mathtt{sid}}$ directly broadcast their pk_i values. For those parties who do not broadcast, the honest or passively corrupted parties provide sufficient information to reconstruct the corresponding pk_i.

As a result, all honest parties derive the same set $\{pk_i = g^{F(i)}\}_{i \in \widetilde{R}_{\mathtt{sid}}}$, and thus compute the same group public key $pk = g^{F(0)}$ and the same public key shares $(pk_i)_{i \in [n]}$.

Finally, note that all the arguments above rely on standard assumptions, including the knowledge soundness and completeness of $\mathsf{NIZK}_{\mathsf{Ped}}$ and $\mathsf{NIZK}_{\mathsf{pk}}$, as well as the Dlog assumption, which underlies the binding property of the Pedersen commitment. An adversary with access to the oracle $\mathcal{O}_{\mathsf{PRF}}$ gains no additional advantage in violating any of these assumptions. Therefore, all the reasoning provided above applies equally to $\mathcal{O}_{\mathsf{PRF}}$-enhanced adversaries. □

Full Secrecy. We prove the concurrent full secrecy of our Anchor-DKG protocol by showing that it satisfies $\mathcal{O}_{\mathsf{PRF}}$-enhanced full secrecy.

Lemma 4 (Full secrecy)**.** *The* Anchor-DKG *protocol in Fig.2 satisfies the $\mathcal{O}_{\mathsf{PRF}}$-enhanced full secrecy. Consequently, it satisfies the concurrent full secrecy as per Def.2.*

Proof. We prove the full secrecy by first building a PPT simulator Sim, which on inputs a uniformly distributed group element $y \in \mathbb{G}$ can simulate an execution where y is group public key. Then, we show that any PPT adversary $\mathcal{A}$, which corrupts up to $t^{\mathsf{rec}}_{\mathtt{sid}} - 1$ parties, with up to $n_{\mathtt{sid}} - t^{\mathsf{rec}}_{\mathtt{sid}}$ Byzantine corruptions, in $\mathcal{P}_{\mathtt{sid}}$, cannot distinguish the simulated execution with a real execution, even if $\mathcal{A}$ has access to $\mathcal{O}_{\mathsf{PRF}}$.

Simulator description. We describe the simulator Sim as follows.

Input: a public key $y \in \mathbb{G}$, generators $g, h \in \mathbb{G}$, the instance information $(\mathtt{sid}, \mathtt{init}, \mathcal{R}_{\mathsf{init}}, \mathcal{N}_{\mathsf{sid}}, t^{\mathsf{rec}}_{\mathtt{sid}}, \vec{T}_{\mathtt{sid}})$, the corruption information $(\mathcal{C}_{\mathtt{B}}, \mathcal{C}_{\mathtt{P}})$, and the master keys of all passively corrupted parties $\{msk_i\}_{i\in\mathcal{C}_{\mathtt{P}}}$.

Random oracle simulation: Sim responds to random oracle queries by following the simulator algorithms of $\mathsf{NIZK}_{\mathsf{Ped}}$ and $\mathsf{NIZK}_{\mathsf{pk}}$.

Preparation: For convenience, let $n = |\mathcal{R}_{\mathsf{init}} \cup \mathcal{N}_{\mathsf{init}}|$. Let $\mathcal{H}_{\mathtt{sid}} \subset [n]$ be the set of honest parties, $\mathcal{C}^{\mathtt{P}}_{\mathtt{sid}} \subset [n]$ the passively corrupted parties, and $\mathcal{C}^{\mathtt{B}}_{\mathtt{sid}} \subset [n]$ the Byzantine parties. Define $\mathcal{C}_{\mathtt{sid}} = \mathcal{C}^{\mathtt{P}}_{\mathtt{sid}} \cup \mathcal{C}^{\mathtt{B}}_{\mathtt{sid}}$.

Simulating round 1: Sim acts on the behalf of P_i for each $i \in \mathcal{H}_{\mathtt{sid}}$ as follows: it uniformly samples $(s_{i,j}, s'_{i,j})$ for all $j \in \mathcal{C}_{\mathtt{sid}}$, and computes $\{\widehat{A}_{i,j}\}_{j\in\mathcal{C}_{\mathtt{sid}}}$. Then, it uniformly samples $\widehat{A}_{i,0} \leftarrow\!\$\ \mathbb{G}$, and interpolates the remaining $\widehat{A}_{i,j}$ for $j \in \mathcal{H}_{\mathtt{sid}}$. It sets $\mathsf{PC}_i = (\widehat{A}_{i,j})_{j\in[0,n]}$.

For $P_i \in \mathcal{C}^{\mathtt{P}}_{\mathtt{sid}}$, Sim honestly executes the protocol throughout the whole protocol and always provides the internal states to $\mathcal{A}$.

Simulating round 2: Sim executes this round honestly on the behalf of all honest nodes, except that honest nodes do not complain each other.

Simulating round 3: Sim executes this round honestly on the behalf of all honest nodes.

Simulating round 4: Sim executes this round honestly on the behalf of all honest nodes until the set $\widetilde{R}_{\mathtt{sid}}$ has been decided. At this stage, Sim knows the following information:

- The secret sharing polynomials $\{\widehat{f}_i, \widehat{f}'_i\}_{i\in\widetilde{R}_{\mathtt{sid}}\cap\mathcal{C}_{\mathtt{sid}}}$.
- The shares from honest or passively corrupted parties to corrupted parties: $\{\widehat{s}_{i,j}, \widehat{s}'_{i,j}\}_{i\in\mathcal{H}_{\mathtt{sid}}\cup\mathcal{C}^{\mathtt{P}}_{\mathtt{sid}}, j\in\mathcal{C}_{\mathtt{sid}}}$.

Then, for the corrupted parties outside $\widetilde{R}_{\mathtt{sid}}$, i.e., $i \in \mathcal{C}_{\mathtt{sid}} \cap ([n] \setminus \widetilde{R}_{\mathtt{sid}})$ it uniformly samples $s_i \leftarrow\!\$\ \mathbb{Z}_p$. Then, based on these information, it interpolates a $(t_{\mathsf{rec}} - 1)$-degree polynomial F in the exponentiation, such that $g^{F(0)} = y$ (the input public key), and $g^{F(i)} = g^{s_i}$ for all $i \in \mathcal{C}_{\mathtt{sid}}$, where $s_i = \widehat{f}_i(0)$ for $i \in \mathcal{C}_{\mathtt{sid}} \cap \widetilde{R}_{\mathtt{sid}}$.

With this polynomial F, it defines $pk_i = g^{F(i)}$ for all $i \in \mathcal{H}_{\mathtt{sid}} \cup \mathcal{C}^{\mathsf{P}}_{\mathtt{sid}}$.

Using the shares $\{\widehat{s}_{i,j}, \widehat{s}'_{i,j}\}_{i \in \mathcal{H}_{\mathtt{sid}} \cup \mathcal{C}^{\mathsf{P}}_{\mathtt{sid}}, j \in \mathcal{C}_{\mathtt{sid}}}$, Sim can interpolate the polynomial f_j for $j \in \mathcal{C}_{\mathtt{sid}}$ such that $f_j(i) = \widehat{s}_{i,j}$.

Note that for any $z \in \mathcal{C}_{\mathtt{sid}} \cap ([n] \setminus \widetilde{R}_{\mathtt{sid}})$, i.e., corrupted parties which are not in $\widetilde{R}_{\mathtt{sid}}$, $F(z)$ and $\{f_j(z)\}_{j \in \mathcal{C}_{\mathtt{sid}}}$ are on the same $(t_{\mathsf{rec}} - 1)$-degree polynomial. This is because $F(z) = \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \lambda_\tau(z) f_\tau(0)$, while $f_j(z) = \sum_{\tau \in \widetilde{R}_{\mathtt{sid}}} \lambda_\tau(z) f_\tau(j)$. Thus, based on $F(z)$ and $\{f_j(z)\}_{j \in \mathcal{C}_{\mathtt{sid}}}$, Sim can interpolate this polynomial (denoted by $\widetilde{f}_z$) and obtain $s_{i,z} = \widetilde{f}_z(i)$ for each $i \in \mathcal{H}_{\mathtt{sid}}$, which is the share supposed to be sent by an honest P_i at Round 5. In addition, Sim can compute the commitment of $s_{i,z}$ as $A_{i,z} = \prod_{\tau \in \widetilde{R}_{\mathtt{sid}}} \widehat{A}_{\tau,i}^{\widehat{\lambda}_\tau(z)}$.

Then, for each $i \in \mathcal{H}_{\mathtt{sid}}$ and $z \in \mathcal{C}_{\mathtt{sid}} \cap ([n] \setminus \widetilde{R}_{\mathtt{sid}})$, Sim runs the simulator algorithm of $\mathsf{NIZK}_{\mathsf{Ped}}$ to generate a simulated proof $\pi_{i,z}$ for $s_{i,z}$ and $A_{i,z}$.

Finally, Sim broadcasts $(\mathtt{sid}, pk_i, \gamma_i)$ on the behalf of honest nodes, while γ_i is a simulated proof. Sim also sends $(\mathtt{sid}, (i, z, \widehat{s}_{i,z}, \pi_{i,z}))$ to each corrupted $z \in \mathcal{C} \cap ([n] \setminus \widetilde{R}_{\mathtt{sid}})$.

Simulating round 5: Sim executes this round honestly on the behalf of all honest nodes.

On the indistinguishability. Now we turn to prove the indistinguishability between $(\mathcal{O}_0, \mathsf{RO}_0)$ simulated by Sim, and $(\mathcal{O}_1, \mathsf{RO}_1)$ which are $\mathcal{O}_{\mathsf{Chall}}$ and a random oracle, respectively. We establish this through the following hybrids.

Hybrid 0: In this hybrid, $\mathcal{A}$ interacts with $\mathcal{O}_{\mathsf{Chall}}$ and a random oracle.

Hybrid 1: It is almost identical to Hybrid 0, except that $\mathcal{O}_{\mathsf{Chall}}$ is replaced with $\mathcal{O}_{H1}$. Compared with the program inside $\mathcal{O}_{\mathsf{Chall}}$, the program inside $\mathcal{O}_{H1}$ takes the following different actions.

- When executing the first round on the behalf of honest repeating parties, $\{P_i\}_{i \in \mathcal{H}_{\mathtt{sid}} \cap \mathcal{R}_{\mathtt{sid}}}$, it does not run $\mathsf{ShareDeriv}(msk_i, \mathtt{sid})$. Instead, it uniformly samples s_i.

From the security of the underlying PRF, $\mathcal{A}$ cannot distinguish Hybrid 1 and Hybrid 0, even when it has access to $\mathcal{O}_{\mathsf{PRF}}$.

Hybrid 2: It is almost identical to Hybrid 1, except that $\mathcal{O}_{H1}$ is replaced with $\mathcal{O}_{H2}$, and RO_1 is replaced with RO_{H2}. Compared with the program inside $\mathcal{O}_{\mathsf{H1}}$, the program inside $\mathcal{O}_{H2}$ takes the following different actions.

- Whenever needs to run $\mathsf{NIZK}_{\mathsf{Ped}}$ for generating proofs for $P_i \in \mathcal{H}_{\mathtt{sid}}$, it invokes the simulator prover algorithm of $\mathsf{NIZK}_{\mathsf{Ped}}$. Meanwhile, the simulator prover algorithm can program the random as needed, which is where RO_{H2} differs from RO_1.

From the zero knowledge property of $\mathsf{NIZK}_{\mathsf{Ped}}$, $\mathcal{A}$ cannot distinguish Hybrid 2 and Hybrid 1, even when it has access to $\mathcal{O}_{\mathsf{PRF}}$.

Hybrid 3: It is almost identical to Hybrid 2, except that $\mathcal{O}_{H2}$ is replaced with $\mathcal{O}_{H3}$, and RO_{H2} is replaced with RO_0. Compared with the program inside $\mathcal{O}_{\mathsf{H2}}$, the program inside $\mathcal{O}_{32}$ takes the following different actions.

- Whenever needs to run $\mathsf{NIZK}_{\mathsf{pk}}$ for generating proofs for $P_i \in \mathcal{H}_{\mathtt{sid}}$, it invokes the simulator prover algorithm of $\mathsf{NIZK}_{\mathsf{pk}}$. Meanwhile, the simulator prover algorithm can program the random as needed, which is where RO_0 differs from RO_{H2}.

Table 1: Comparison of DKG protocols with full secrecy

	Share derivation?	**Setup**†	**Robust?**	**Rounds**‡	**Broadcast cost**∗	**Comp. efficiency**∗∗
GJKR [35] (optimized)	✗	Sec. Chan.	✓	4 Broad.	$O(n^2\lambda)$	★★★
Katz-I [39, Sect. 5]	✗	PKI	✓	2 Broad.	$O(n^2\lambda)$	★★
Katz-II [39, Sect. 6]	✗	Sec. Chan.	✓	2 Broad.	$O\left(n^2\lambda\binom{n}{t}\right)$	★
Katz-III [39, Sect. 7]	✗	1 Rd. Prep. & Sec. Chan.	✓	2 Broad.	$O\left(n\lambda\binom{n}{t}\right)$	★★★
Katz-IV [39, Sect. 8]	✗	2 Rd. Prep. & Sec. Chan.	✓	1 Broad.	$O\left(n\lambda\binom{n}{t}\right)$	★★★
Komlo et al. [41]	✗	Sec. Chan.	✗	3 Broad.	$O(n^2\lambda)$	★★★
Ours	✓	Sec. Chan.	✓	4 Broad. +1 P2P	$O(n^2\lambda)$	★★★

† "x Rd. Prep." denotes x rounds of preprocessing. We omit the random oracle as a setup since, in practice, it is instantiated with a fixed hash function. We also omit the CRS when it can be replaced by a random oracle. "Sec. Chan." denotes secure channels that are both private and authenticated.

‡ "x Broad. $+y$ P2P" denotes x broadcast rounds and y additional point-to-point rounds. The number of rounds is measured in the adversarial case.

∗ Number of bits sent by honest parties via the broadcast channel.

∗∗ Katz-I uses NIZK proofs for the well-formedness of encrypted shares, which are computationally expensive, as evaluated in [36]. Katz-II relies on heavy machinery such as multiparty non-interactive key exchange [12].

From the zero knowledge property of $\mathsf{NIZK}_{\mathsf{pk}}$, $\mathcal{A}$ cannot distinguish Hybrid 3 and Hybrid 2, even when it has access to $\mathcal{O}_{\mathsf{PRF}}$.

Hybrid 4: In this Hybrid, $\mathcal{A}$ interacts with $\mathcal{O}_0$ and RO_0.

It is easy to verify that this Hybrid is identical to Hybrid 3 in the view of $\mathcal{A}$. Consequently, Hybrid 4 and Hybrid 3 are indistinguishable to $\mathcal{A}$. □

5 Evaluation

We evaluate the communication and computation costs.

Comparison with Other Full-Secrecy Protocols. Table 1 compares our Anchor-DKG with existing DKG protocols with *full secrecy*. Despite recent progress, only a few protocols achieve full secrecy, which guarantees unbiased keys, as noted in [39, 40].

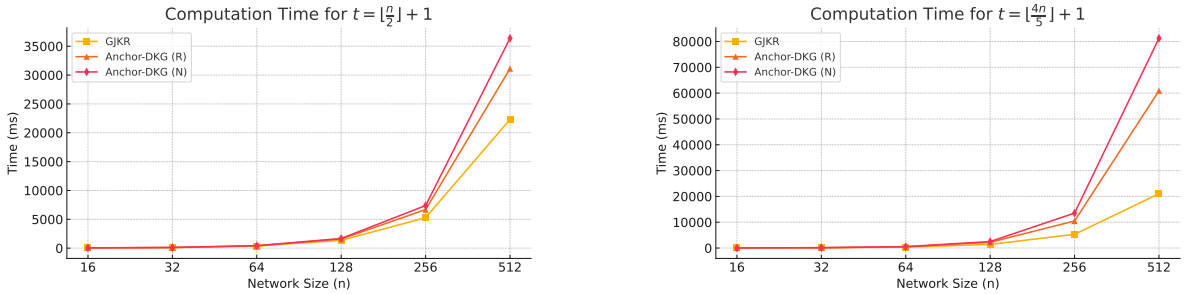
Note that although several recent synchronous DKG designs, such as Dragon-DKG [28], Grand-DKG [9], and Sync-DKG [48], do not assume broadcast channels, none of them have been implemented in practice, largely due to the complexity of the consensus mechanisms they rely on. For instance, Dragon-DKG [28] and Grand-DKG [9] employ deterministic Byzantine agreement, which incurs $O(n)$ round complexity, while the randomized consensus used in Sync-DKG [48] requires a few tens of rounds in expectation. While designing an efficient Anchor-DKG protocol without assuming broadcast channels is an interesting direction, we focus in this paper on the more traditional broadcast setting, as a first step toward addressing key-management challenges in complex

threshold cryptosystems and distilling the core design ideas.

As shown in Table 1, considering both computational efficiency and broadcast cost, the GJKR variant [35][9] remains one of the most practical candidates despite requiring more broadcast rounds than some recent designs. It also shares the same setup as ours. Therefore, we select it as our baseline, and compare our protocol against it in concrete communication and computation costs, with the aim of showing that Anchor-DKG introduces only minimal overhead relative to a conventional DKG without share derivation.

Configurations. Our protocol involves the following basic operations: elliptic curve cryptographic (ECC) group operations and hashing to the scalar field of the ECC group. For our performance evaluation, we adopt `secp256k1` as the ECC group and use `SHA-256` as the underlying hash function.

We implemented both the baseline protocol and our Anchor-DKG as round functions in Java, built upon the mpc4j library [10]. In addition, the ECC and hash operations are supported by the BouncyCastle library[11], while the polynomial operations are handled using the Redberry library[12].



(a) Computation time for $t^{\mathsf{rec}} = \lfloor \frac{n}{2} \rfloor + 1$

(b) Computation time for $t^{\mathsf{rec}} = \lfloor \frac{4n}{5} \rfloor + 1$

Figure 3: Computation Comparison of GJKR vs. Anchor-DKG (repeating vs. non-repeating nodes)

Communication Performance. We calculate communication costs with respect to the broadcast channel (BC) and point-to-point (p2p) channels, measured as the total size of messages sent by *all* honest parties. Table 2 summarizes the communication costs of Anchor-DKG and GJKR in the optimistic case where no malicious participants are present.

GJKR and our Anchor-DKG share nearly identical first three rounds and both employ the same technique for extracting the group public key. This results in identical communication costs over the broadcast channel. In Round 4, parties in Anchor-DKG additionally send private helper information to parties not in $\widetilde{R}_{\mathtt{sid}}$ for computing their secret key shares, explaining the around 75% cost increase in p2p channel costs.

We remark that in adversarial settings, both protocols can incur significantly higher communication costs. The increase in Anchor-DKG may be slightly larger due to the opening messages in Round 5.

Regarding round complexity, note that both protocols require 4 broadcast rounds in the optimistic case (no messages are sent in the second and third rounds). In the adversarial case,

[9] See Appendix B for details. Briefly, we apply techniques from [39] to reduce the original GJKR protocol by two broadcast rounds.

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

[11] `https://www.bouncycastle.org/`

[12] `http://redberry.cc/`

Table 2: Communication-cost comparison between GJKR and Ours. $|g|$ stands for the size of $g \in \mathbb{G}$, and $t^{\mathsf{rec}} = \lfloor \frac{n}{2} \rfloor + 1$.

	GJKR/Ours-BC	**GJKR-p2p**	**Ours-p2p**
n	$(n^2 + 2n)\|g\| + 3n \log q$	$2n^2 \log q$	$3.5n^2 \log q$
32	38.06 KiB	64 KiB	112 KiB
128	548.31 KiB	1 MiB	1.75 MiB
512	8.33 MiB	16 MiB	28 MiB

Anchor-DKG requires one additional p2p round.

Computation Performance. In Fig. 3, we compare the per-node running time of our protocol and GJKR for network sizes ranging from $n = 16$ to $n = 512$, under two reconstruction thresholds: $t^{\mathsf{rec}} = \lfloor n/2 \rfloor + 1$ and $t^{\mathsf{rec}} = \lfloor 4n/5 \rfloor + 1$ (see Fig. 3a and Fig. 3b). The reported times include local computation for verifying all incoming messages and generating outgoing messages, but exclude network transmission and reception delays. Since parties in $\widetilde{R}_{\mathtt{sid}}$ and those outside $\widetilde{R}_{\mathtt{sid}}$ perform different tasks in the final two rounds of Anchor-DKG, we report their computation costs separately.

Due to extra operations needed to assist parties outside $\widetilde{R}_{\mathtt{sid}}$, and the additional verifications, their costs are consistently higher than those costs in GJKR. These overheads include $O(n^2)$ group exponentiations, explaining the growing gap as n increases. For instance, when $n = 128$, $t^{\mathsf{rec}} = 65$, the cost for $P_i \in \widetilde{R}_{\mathtt{sid}}$ is 1.59 s vs. 1.36 s in GJKR, and when $n = 512$, the gap widens to 31 s vs. 22.3 s.

Acknowledgment

We thank anonymous reviewers of CCS'26 for valuable suggestions. This work is supported in part by ARC Discovery Project (DP250101739), and research awards from the Stellar Development Foundation, SUI academic awards and Input Output (iog.io) as part of the Cardano Vision 2026 research initiative.

Ethical Considerations

Our work improves the management of secret-key shares across threshold cryptosystems, reducing the risk of key loss or compromise. Although cryptographic techniques may be misused, Anchor Distributed Key Generationis intended to strengthen security and privacy in decentralized systems. We will responsibly disclose any vulnerabilities identified in the protocol or its implementation and make our results openly available to support broad and equitable access.

Open Science

Our entire artifact is openly available at:

`https://github.com/Hanwen-Feng/Anchor-DKG-Artifact-Evaluation`

The artifact provides implementations of the round functions used in our Anchor DKG protocol, together with implementations of the corresponding round functions of the baseline GJKR-DKG protocol. It also includes benchmarking code for measuring the local execution time of these round functions under different system scales and parameter settings. Using the provided scripts and configurations, users can reproduce the experimental results reported in Fig.3.

References

[1] `https://www.zama.ai/products-and-services/fhe-threshold-key-management-system`.

[2] `https://eth2book.info/latest/part2/building_blocks/signatures/`.

[3] Blockchain oracles. `https://chain.link/education/blockchain-oracles`.

[4] Eigenlayer: The restaking collective. `https://docs.eigenlayer.xyz/assets/files/EigenLayer_WhitePaper-88c47923ca0319870c611decd6e562ad.pdf`.

[5] Multi-party threshold cryptography. *NIST Computer Security Resource Center*, 2024. `https://csrc.nist.gov/projects/threshold-cryptography`.

[6] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 39–70. Springer, Cham, August 2023.

[7] Will Arthur, David Challener, and Kenneth Goldman. *A practical guide to TPM 2.0: Using the new trusted platform module in the new age of security*. Springer Nature, 2015.

[8] Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. Network-agnostic security comes (almost) for free in DKG and MPC. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 71–106. Springer, Cham, August 2023.

[9] Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. Grandline: Adaptively secure DKG and randomness beacon with (almost) quadratic communication complexity. *IACR Cryptol. ePrint Arch.*, page 1887, 2023.

[10] Renas Bacho and Julian Loss. On the adaptive security of the threshold BLS signature scheme. In *CCS*, pages 193–207. ACM, 2022.

[11] Leemon Baird, Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Threshold signatures in the multiverse. 2023. IEEE Symposium on Security and Privacy (SP).

[12] Dan Boneh, Darren Glass, Daniel Krashen, Kristin Lauter, Shahed Sharif, Alice Silverberg, Mehdi Tibouchi, and Mark Zhandry. Multiparty non-interactive key exchange and more from isogenies on elliptic curves. *Journal of Mathematical Cryptology*, 14(1):5–14, 2020.

[13] Jan Bormet, Sebastian Faust, Hussien Othman, and Ziyan Qu. BEAT-MEV: Epochless approach to batched threshold encryption for MEV prevention. Cryptology ePrint Archive, Paper 2024/1533, 2024.

[14] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[15] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *CRYPTO*, volume 1666 of *LNCS*, pages 98–115. Springer, 1999.

[16] Ran Canetti and Tal Rabin. Universal composition with joint state. In *Advances in Cryptology-CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings 23*, pages 265–281. Springer, 2003.

[17] Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *ACNS*, volume 10355 of *LNCS*, pages 537–556. Springer, 2017.

[18] Colin Roberts and Vivek Nair and Dawn Song. Wrangling entropy: Next-generation multi-factor key derivation, credential hashing, and credential generation functions, 2025.

[19] Elizabeth Crites, Chelsea Komlo, and Mary Maller. Fully adaptive schnorr threshold signatures. In *annual international cryptology conference*, pages 678–709. Springer, 2023.

[20] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. Verifiable secret sharing simplified. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 46–46. IEEE Computer Society, 2024.

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

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

[23] Yevgeniy Dodis. Efficient construction of (distributed) verifiable random functions. In Yvo Desmedt, editor, *PKC 2003*, volume 2567 of *LNCS*, pages 1–17. Springer, Berlin, Heidelberg, January 2003.

[24] Elaine Barker and Lily Chen. Recommendation for Key Derivation Using Pseudorandom Functions. Technical Report NIST Special Publication 800-108, National Institute of Standards and Technology, October 2009.

[25] Elaine Barker, Lily Chen, and Allen Roginsky. Recommendation for Key-Derivation Methods in Key-Establishment Schemes. Technical Report NIST Special Publication 800-56C Rev. 2, National Institute of Standards and Technology, January 2020.

[26] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for Key Management, Part 1: General. Technical Report NIST Special Publication 800-57 Part 1 Rev. 5, National Institute of Standards and Technology, May 2020.

[27] Thomas Espitau, Guilhem Niot, and Thomas Prest. Flood and submerse: Distributed key generation and robust threshold signature from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 425–458. Springer, Cham, August 2024.

[28] 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 Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *43rd ACM PODC*, pages 469–479. ACM, June 2024.

[29] Hanwen Feng, Tiancheng Mai, and Qiang Tang. Scalable and adaptively secure any-trust distributed key generation and all-hands checkpointing. In Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 2636–2650. ACM Press, October 2024.

[30] Hanwen Feng and Qiang Tang. Asymptotically optimal adaptive asynchronous common coin and dkg with silent setup. In *Annual International Cryptology Conference*, pages 642–675. Springer, 2025.

[31] Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In *Public Key Cryptography*, volume 1992 of *LNCS*, pages 300–316. Springer, 2001.

[32] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: MPC, encryption and signatures. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 295–327. Springer, Cham, August 2023.

[33] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. hints: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3034–3052, 2024.

[34] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO (7)*, volume 14926 of *Lecture Notes in Computer Science*, pages 352–386. Springer, 2024.

[35] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83, 2007.

[36] Craig Gentry, Shai Halevi, and Vadim Lyubashevsky. Practical non-interactive publicly verifiable secret sharing with thousands of parties. In *EUROCRYPT (1)*, volume 13275 of *LNCS*, pages 458–487. Springer, 2022.

[37] Jens Groth and Victor Shoup. Fast batched asynchronous distributed key generation. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 370–400. Springer, Cham, May 2024.

[38] 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 Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie, editors, *ACM CCS 2024*, pages 4286–4300. ACM Press, October 2024.

[39] Jonathan Katz. Round-optimal, fully secure distributed key generation. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 285–316. Springer, Cham, August 2024.

[40] Chelsea Komlo and Ian Goldberg. FROST: Flexible round-optimized Schnorr threshold signatures. In Orr Dunkelman, Michael J. Jacobson, Jr., and Colin O'Flynn, editors, *SAC 2020*, volume 12804 of *LNCS*, pages 34–65. Springer, Cham, October 2020.

[41] Chelsea Komlo, Ian Goldberg, and Douglas Stebila. A formal treatment of distributed key generation, and new constructions. Cryptology ePrint Archive, Report 2023/292, 2023.

[42] Chelsea Komlo, Ian Goldberg, and Christopher A. Wood. The Flexible Round-Optimized Schnorr Threshold (FROST) Signing Protocol. RFC 9591, June 2024. `https://doi.org/10.17487/RFC9591`.

[43] Hugo Krawczyk and Pasi Eronen. Hmac-based extract-and-expand key derivation function (hkdf). `https://tools.ietf.org/html/rfc5869`, 2010. RFC 5869.

[44] Vivek Nair and Dawn Song. Multi-factor key derivation function (MFKDF) for fast, flexible, secure, & practical key management. In Joseph A. Calandrino and Carmela Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 2097–2114. USENIX Association, 2023.

[45] National Institute of Standards and Technology. Security Requirements for Cryptographic Modules. Technical Report FIPS PUB 140-3, U.S. Department of Commerce, March 2019.

[46] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576 of *LNCS*, pages 129–140. Springer, 1991.

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

[48] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. Cryptology ePrint Archive, Paper 2021/1635, 2023. `https://eprint.iacr.org/2021/1635`.

[49] Yaqing Song, Yuan Zhang, Shiyu Li, Weijia Li, Zeqi Lai, and Qiang Tang. Non-interactive distributed key management with pre-determined shares. In *46th IEEE International Conference on Distributed Computing Systems, ICDCS 2026, Seoul, South Korea, June 22-25, 2026*. IEEE, 2026.

[50] Andrei Tonkikh and Luciano Freitas de Souza. Swiper: a new paradigm for efficient weighted distributed protocols. In *PODC*, pages 283–294. ACM, 2024.

[51] Tiancheng Xie, Jiaheng Zhang, Zerui Cheng, Fan Zhang, Yupeng Zhang, Yongzheng Jia, Dan Boneh, and Dawn Song. zkbridge: Trustless cross-chain bridges made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 3003–3017, New York, NY, USA, 2022. Association for Computing Machinery.

[52] Adam L Young and Moti Yung. Cryptovirology: The birth, neglect, and explosion of ransomware. *Communications of the ACM*, 60(7):24–26, 2017.

[53] Alexey Zamyatin, Konstantinos Chalkias, Nikos Fotiou, and Andreas Tikkas. Hierarchical deterministic key derivation for bls signatures. In *Proceedings of the IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 304–315. IEEE, 2022.

A Supplementary Material to §4

A.1 Building Blocks

NIZKPoK Definitions. A NIZKPoK for an NP relation Rela in the random oracle model can be described by the following two algorithms, where RO represents the random oracle(s) needed by the algorithm.

- $\mathsf{Prove}^{\mathsf{RO}}(X, W)$ on inputs a statement X and its witness W such that $(X, W) \in \mathsf{Rela}$, produces a proof π.
- $\mathsf{Vrfy}^{\mathsf{RO}}(X, \pi)$ validates the proof π for X.

We consider the following properties of NIZKPoK

- *Completeness.* For any $(X, W) \in \mathsf{Rela}$, it holds that

$$\Pr[\mathsf{Vrfy}^{\mathsf{RO}}(X, \mathsf{Prove}^{\mathsf{RO}}(X, W)) = 1] = 1$$

.

- *Knowledge Soundness.* For any PPT adversary $\mathcal{A}$, there is an PPT algorithm $E_{\mathcal{A}}$, such that

$$\Pr\begin{bmatrix}(X,\pi)\leftarrow\mathcal{A}^{\mathsf{RO}}(1^\lambda), W\leftarrow E_{\mathcal{A}}(X):\\ \mathsf{Vrfy}(X,\pi)=1\wedge(X,W)\notin\mathsf{Rela}\end{bmatrix}\leq\mathsf{negl}(\lambda).$$

- *Zero Knowledge.* If there is a simulator $\mathsf{SimProve}$, and for any PPT adversary $\mathcal{A}$, it holds that

$$\left|\begin{matrix}|\Pr[1\leftarrow\mathcal{A}^{\mathcal{O}_0(\cdot),\mathsf{RO}}(1^\lambda)]-\\ \Pr\left[1\leftarrow\mathcal{A}^{\mathcal{O}_1(\cdot),\mathsf{Sim}_{\mathsf{RO}}}(1^\lambda)\right]\end{matrix}\right|\leq\mathsf{negl}(\lambda),$$

  where $\mathcal{O}_0$ on input $(X,W)\in\mathsf{Rela}$ returns $\pi\leftarrow\mathsf{Prove}^{\mathsf{RO}}(X,W)$, and $\mathcal{O}_1$ returns $\pi\leftarrow\mathsf{SimProve}(x)$, while $\mathsf{Sim}_{\mathsf{RO}}$ is handled by $\mathsf{SimProve}$ as well.

NIZKPoK Constructions. We recall the details of NIZK_{Ped} and $\mathsf{NIZK}_{\mathsf{pk}}$ in the following, which are classical constructions obtained by applying Fiat-Shamir to Sigma protocols.

$\mathsf{NIZK}_{\mathsf{Ped}}$ *construction.* For the relation $\mathsf{Rela}_{\mathsf{Ped}}$ (cf. Eq.6), $\mathsf{NIZK}_{\mathsf{Ped}}$ can be constructed as follows.

- $\mathsf{NIZK}_{\mathsf{Ped}}.\mathsf{Prove}^{\mathsf{RO}}((C,g,h,s),s')$. It samples $r\leftarrow\!\!\$\,\mathbb{Z}_q$, obtains $ch=\mathsf{RO}((C,g,h,s),h^r)$, and computes $z=r+ch\cdot s' \mod q$. It returns $\pi=(ch,z)$.
- $\mathsf{NIZK}_{\mathsf{Ped}}.\mathsf{Vrfy}^{\mathsf{RO}}((C,g,h,s),(ch,z))$. It computes $H=C/(g^s)$, and obtains $ch'=\mathsf{RO}((C,g,h,s),h^z/H^{ch})$. It returns 1 if $ch'=ch$, or it returns 0 otherwise.

$\mathsf{NIZK}_{\mathsf{pk}}$ *construction.* For the relation $\mathsf{Rela}_{\mathsf{pk}}$ (cf. Eq.7), $\mathsf{NIZK}_{\mathsf{pk}}$ can be constructed as follows.

- $\mathsf{NIZK}_{\mathsf{pk}}.\mathsf{Prove}^{\mathsf{RO}}((C,g,h,pk),(s,s'))$. It samples $r,r'\leftarrow\!\!\$\,\mathbb{Z}_q$, obtains $ch=\mathsf{RO}((C,g,h,pk),(g^r,h^{r'}))$, and computes $z=r+ch\cdot s \mod q, z'=r'+ch\cdot s'$. It returns $\pi=(ch,z,z')$.
- $\mathsf{NIZK}_{\mathsf{pk}}.\mathsf{Vrfy}^{\mathsf{RO}}((C,g,h,pk),(ch,z,z'))$. It computes $H=C/pk$, and obtains $ch'=\mathsf{RO}((C,g,h,pk),(g^z/pk^{ch},h^{z'}/H^{ch}))$. It returns 1 if $ch'=ch$, or it returns 0 otherwise.

A.2 Discussion on Adaptive Security

Recall that in our model, the adversary specifies the corruption status of each party upon its introduction to the system. A stronger and more flexible model, known as adaptive security, would allow the adversary to corrupt any party at any point during the protocol execution. Although we are not aware of any adaptive attack that breaks the robustness or secrecy of our protocol, there are significant barriers to providing a formal proof of adaptive security. Specifically, we encounter the following "commitment" issues inherent to simulation-based proofs:

- **Commitment to shares.** The simulator must commit to the shares of honest parties without knowing their underlying values (as the final secret key sk must remain unknown to the simulator). If an honest party is later corrupted, the simulator must open these commitments to values consistent with the prior execution.

- **Consistency of proofs.** To achieve full secrecy, the protocol reconstructs the public key pk in the final step, requiring each party P_i to publish its public key share pk_i along with a validity proof. In the simulation, the simulator must generate a simulated proof γ_i for the honest party's pk_i because the corresponding secret share is unknown. However, upon adaptive corruption, the simulator is required to reveal the internal randomness and secret witness consistent with both pk_i and the simulated proof γ_i, a task known to be difficult for standard proof systems.
- **Consistency across committees.** Additionally, the simulator must ensure that the shares of parties recurring across different committees appear as valid PRF evaluations under a single master key, all while the master keys of honest nodes are hidden to the simulator.

Indeed, all existing DKG schemes with full secrecy and public shares of the form $pk_i = g^{sk_i}$ are limited to static security. Our security model therefore matches the state of the art. While some recent schemes (e.g., [10]) achieve adaptive security, they sacrifice full secrecy and rely on strong assumptions such as the algebraic group model. Designing a DKG that is both adaptively secure and fully secure remains an important open problem; its resolution likely requires orthogonal techniques beyond the scope of this work.

A.3 Deployment Discussion

Anchor-DKG is designed for environments where participants exhibit diverse lifecycles: while some nodes are transient, others naturally persist across many committee instantiations. A prime example is a decentralized oracle network (DON) (e.g., [12]), where clients dynamically select committees to attest to off-chain data. In such settings, high-performing nodes cultivate strong reputations and are frequently selected. These parties have a strong incentive to minimize state bloat and can designate themselves as repeating nodes, leveraging our master-key derivation to maintain constant-size storage regardless of the number of committees they serve. Conversely, newer or less frequent participants act as transient nodes, accepting linear storage overhead in exchange for the opportunity to build reputation.

When forming a committee, a client specifies a parameter t_{rec} and includes at most t_{rec} repeating nodes. This choice represents a strategic trade-off: increasing t_{rec} generally enhances privacy (by raising the corruption threshold) and network efficiency (by allowing more nodes to use lightweight storage), though it may theoretically reduce liveness robustness if the threshold becomes too high. However, since repeating nodes are selected precisely for their proven stability and reputation, clients can often safely maximize t_{rec} to fully exploit the efficiency benefits of the protocol.

B An Optimized Variant of GJKR

The first three rounds of this variant are identical to the original protocol, where each dealer shares a random secret via Pedersen VSS and disputes are resolved through complaints. After agreeing on a qualified set Qual of valid VSS instances, instead of proceeding with two broadcast rounds and one point-to-point round to extract the group public key pk (as in the original), the variant applies a standard optimization—also seen in [39]—that reduces this to a single broadcast round: each node simply broadcasts its public key share along with a NIZK proof (as in Line 21 of Round 4 in our protocol).